IRIX User's Reference Manual

Volume II

Version 5.0

Document Number 007-0606-050

5/90

IRIX User's
Reference Manual
Version 5.0
Document Number 007-0606-050

Silicon Graphics, Inc.
Mountain View, California

TABLE OF CONTENTS

1. Commands

NAME

intro – introduction to commands, application programs, and programming commands.

DESCRIPTION

This section describes, in alphabetical order, publicly-accessible commands. Certain distinctions of purpose are made in the headings:

(1)	Commands of general utility.
(1C)	Commands for communication with other systems.
(1G)	Graphics utilities.

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [*–option* ...] [*cmdarg* ...]
where:

[ ] Surround an *option* or *cmdarg* that is not required.

... Indicates multiple occurrences of the *option* or *cmdarg*.

name The name of an executable file.

option This is either
noargletter ...
or
argletter optarg [,...]
It is always preceded by a "–".

noargletter
A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "–" (Rule 5, below).

argletter
A single letter representing an option requiring an option-argument.

optarg
An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by white space and quoted (Rule 8, below).

cmdarg
Path name (or other command argument) *not* beginning with "–", or "–" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. *getopts*(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lower-case letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "–".
5. Options with no arguments may be grouped after a single "–".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., `-o xxx,z,yy` or `-o "xxx z yy"`).
9. All options must precede operands (*cmdarg* above) on the command line.
10. "– –" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "–" preceded and followed by white space should only be used to mean standard input.

Throughout the manual pages there are references to *TMPDIR*, *BINDIR*, *INCDIR*, *LIBDIR*, and *LLIBDIR*. These represent directory names whose value is specified on each manual page as necessary. For example, *TMPDIR* might refer to /tmp or /usr/tmp. These are not environment variables and cannot be set. (There is also an environment variable called **TMPDIR** which can be set. See *tmpnam*(3S).)

SEE ALSO

getopts(1), exit(2), wait(2), getopt(3C).

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of ''normal'' termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, or bad or inaccessible data. It is called variously ''exit code'', ''exit status'', or ''return code'', and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

M – N

NAME

m4 – macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

The *m4* command is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is –, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

–e	Operate interactively. Interrupts are ignored and the output is unbuffered.
–s	Enable line sync output for the C preprocessor (#line . . .)
–B*int*	Change the size of the push-back and argument collection buffers from the default of 4,096.
–H*int*	Change the size of the symbol table hash array from the default of 199. The size should be prime.
–S*int*	Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
–T*int*	Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **–D** or **–U** flags:

–D*name*[=*val*]	Defines *name* to *val* or to null in *val*'s absence.
–U*name*	Undefines *name*.

Macro calls have the form:

```
name(arg1, arg2, . . . , argn)
```

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define	the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotes).
undefine	removes the definition of the macro named in its argument.
defn	returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	like *define*, but saves any previous definition.
popdef	removes current definition of its argument(s), exposing the previous one, if any.
ifdef	if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX system versions of *m4*.
shift	returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
changequote	change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., \` ´).

changecom	change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	*m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next new-line.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	returns the value of its argument decremented by 1.
eval	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to *include*, except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX system command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup()´)
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).

NAME

machid: pdp11, u3b, u3b2, u3b5, u3b15, vax, m68k, m68000, mips, 4d, 4d60 – get processor type truth value

SYNOPSIS

pdp11

u3b

u3b2

u3b5

u3b15

vax

m68k

m68000

mips

4d

4d60

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

pdp11	True if you are on a PDP-11/45 or PDP-11/70.
u3b	True if you are on a 3B20 computer.
u3b2	True if you are on a 3B2 computer.
u3b5	True if you are on a 3B5 computer.
u3b15	True if you are on a 3B15 computer.
vax	True if you are on a VAX-11/750 or VAX-11/780.
4d	True if you are on an IRIS-4D series workstation.
4d60	True if you are on an IRIS-4D series workstation (obsolete, use 4d).
m68k	True if you are on a IRIS-3000 series workstation.
mips	True if you are on a MIPS R2000 or R3000 microprocessor.
m68000	True if you are on a 68000.

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles (see *make*(1)) and shell procedures (see *sh*(1) and *csh*(1)) to increase portability.

SEE ALSO

csh(1), sh(1), test(1), true(1).
make(1) in the *Programmer's Reference Manual.*

NAME

mail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**–wt**] persons

Reading mail:

mail [**–ehpqr**] [**–f** file] [**–F** persons]

DESCRIPTION

Sending mail:

The command-line arguments that follow affect SENDING mail:

–w	causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.
–t	causes a **To:** line to be added to the letter, showing the intended recipients.

A *person* is usually a user name recognized by *login*(1). When *persons* are named, *mail* assumes a message is being sent (except in the case of the –F option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, *mail* adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. **dead.letter** is recreated every time it is needed, erasing any previous contents.

Mail may be sent to a recipient on a remote system. Replace *person* with a UUCP route such as 'host1!host2!user' to send mail thru the UUCP network connect to this system. Use an RFC-822 address such as 'user@host.domain' to allow *sendmail* to figure out what to do with it.

Reading Mail:

The command-line arguments that follow affect READING mail:

–e	causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

−h	causes a window of headers to be displayed rather than the latest message. The display is followed by the '**?**' prompt.
−p	causes all messages to be printed without prompting for disposition.
−q	causes *mail* to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.
−r	causes messages to be printed in first-in, first-out order.
−f*file*	causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.
−F*persons*	entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*. See also *.forward* files and *sendmail(1M).*

mail, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<new-line>, **+**, or **n**	Go on to next message.
d, or **dp**	Delete message and go on to next message.
d #	Delete message number #. Do not go on to next message.
dq	Delete message and quit *mail.*
h	Display a window of headers around current message.
h #	Display header of message number #.
h a	Display headers of ALL messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
p	Print current message again.
−	Print previous message.
a	Print message that arrived during the *mail* session.
#	Print message number #.
r [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *files* (**mbox** is default).

y	Same as save.
u [#]	Undelete message number # (default is last read).
w [*files*]	Save message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons*.
q, or **ctl-d**	Put undeleted mail back in the *mailfile* and quit *mail.*
x	Put all mail back in the *mailfile* unchanged and exit *mail.*
!*command*	Escape to the shell to do *command*.
?	Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **–F** option.

To forward all of one's mail to systema!user enter:

```
mail –Fsystema!user
```

To forward to more than one user enter:

```
mail –F"user1,systema!user2,systema!systemb!user3"
```

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the –**F** option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

To remove forwarding enter:

```
mail -F ""
```

The pair of double quotes is mandatory to set a NULL argument for the -F option.

In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

.forward files understood by *sendmail* provide another, slightly easier to use mechanism to forward mail.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mail_bsd(1), rmail(1M), sendmail(1M), uux(1C), write(1).
User's Guide.
System Administrator's Guide.

WARNING

The "Forward to person" feature may result in a loop, if *sys1!userb* forwards to *sys2!userb* and *sys2!userb* forwards to *sys1!userb*. The symptom is a message saying "unbounded...saved mail in dead.letter."

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

Mail – send and receive mail

SYNOPSIS

Mail [**–v**] [**–i**] [**–n**] [–s *subject*] [*user...*]
Mail [**–v**] [**–i**] [**–n**] **–f** [*name*]
Mail [**–v**] [**–i**] [**–n**] **–u** *user*

INTRODUCTION

Mail is a intelligent mail processing system, which has a command syntax reminiscent of *ed* with lines replaced by messages.

The **–v** flag puts *Mail* into verbose mode; the details of delivery are displayed on the users terminal. The **–i** flag causes tty interrupt signals to be ignored. This is particularly useful when using *Mail* on noisy phone lines. The **–n** flag inhibits the reading of /usr/lib/Mail.rc.

Sending mail. To send a message to one or more other people, *Mail* can be invoked with arguments which are the names of people to send to. You are then expected to type in your message, followed by an EOT (control–D) at the beginning of a line. A subject may be specified on the command line by using the –s flag. (Only the first argument after the –s flag is used as a subject; be careful to quote subjects containing spaces.) The section below, labeled *Replying to or originating mail,* describes some features of *Mail* available to help you compose your letter.

Reading mail. In normal usage *Mail* is given no arguments and checks your mail out of the post office, then prints out a one line header of each message there. The current message is initially the first message (numbered 1) and can be printed using the **print** command (which can be abbreviated p). You can move among the messages much as you move between lines in *ed*, with the commands '+' and '–' moving backwards and forwards, and simple numbers.

Disposing of mail. After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the *Mail* program to forget about the message. This is not irreversible; the message can be **undeleted** (**u**) by giving its number, or the *Mail* session can be aborted by giving the **exit** (**x**) command. Deleted messages will, however, usually disappear never to be seen again.

Specifying messages. Commands such as **print** and **delete** can be given a list of messages as an argument in order to apply to a number of messages at once. This list of messages can be specified in one of three mutually exclusive ways:

First, messages may be specified by message number. Thus ``delete 1 2'' deletes messages 1 and 2, while ``delete 1–5'' deletes messages 1 through 5. In conjunction with message numbers, the following special names may also be used: The special name '^' addresses the first message, and '$' addresses the last message.

Second, messages may be specified by sender's name, sender's name substring or subject substring. You may supply a list of whitespace separated message senders' names and/or message senders' names substrings and/or message subject substrings to commands accepting message lists. A message sender's name is a string of characters which must begin with an alphabetic character, and must match exactly the sender's name in the target message. A message sender's name substring is a '?' character immediately followed (no whitespace) by a string of characters and specifies all messages with a sender's name containing the character string as a substring. A message subject is a '/' character immediately followed (no whitespace) by a string of characters and specifies all messages with a subject line containing the character string as a substring. Examples: ``delete foo'' deletes all messages from ``foo'' exactly, while ``delete /foo'' deletes all messages with substring ``foo'' in their subject lines and ``delete ?foo'' deletes all messages with substring foo in their senders' names.

Third, the special name '*' can be used to address all messages. Thus the command **top** which prints the first few lines of a message could be used in ``top *'' to print the first few lines of all messages.

Replying to or originating mail. You can use the **reply** command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to an end-of-file, defines the contents of the message. While you are composing a message, *Mail* treats lines beginning with the character '~' specially. For instance, typing ``~m'' (alone on a line) will place a copy of the current message into the response right shifting it by a tabstop. Other escapes will set up subject fields, add and delete recipients to the message and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options are given in the summary below.)

Ending a mail processing session. You can end a *Mail* session with the **quit (q)** command. Messages which have been examined go to your *mbox* file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. The **–f** option causes *Mail* to read in the contents of your *mbox* (or the specified file) for processing; when you **quit**, *Mail* writes undeleted messages back to this file. The –u flag is a short way of doing "Mail –f /usr/mail/user".

Personal and systemwide distribution lists. It is also possible to create a personal distribution lists so that, for instance, you can send mail to ''cohorts'' and have it go to a group of people. Such lists can be defined by placing a line like

```
alias cohorts bill ozalp jkf mark kridle@ucbcory
```

in the file .mailrc in your home directory. The current list of such aliases can be displayed with the alias (a) command in *Mail*. System wide distribution lists can be created by editing /usr/lib/Mail.rc (which may contain other *Mail* commands such as set). An alias of the form

```
alias bob sauron!bob
```

will be ignored on the sauron system so that the same /usr/lib/Mail.rc or ~/.mailrc file may be used on several machines with correct behavior.

A signature line (or lines) may be automatically appended to the end of all outgoing letters by placing the text in the file .lsignature, .rsignature, or .signature in your home directory. The file .lsignature is used for local mail, that is the recipients specified do not have '!' or '@' in their names (prior to aliasing) and the file .rsignature is used for remote mail. If the appropriate one of these does not exist, .signature is used for compatibility with previous versions of *Mail*.

Mail has a number of options which can be set in the *.mailrc* file to alter its behavior; thus ''set askcc'' enables the ''askcc'' feature. (These options are summarized below.)

SUMMARY

(Adapted from the 'Mail Reference Manual')

Each command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety – the first command which matches the typed prefix is used. For commands which take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, *Mail* types ''No applicable messages'' and aborts the command.

!	Executes the UNIX shell command which follows.
–	Goes to the previous message and prints it out.
=	Prints the current message number.
?	Prints a brief summary of commands.

More	(M) Like **Print** but invokes your pager.
New	(N) Identical to the **unread** command.
Page	(Pa) A synonym for More.
Print	(P) Like **print** but also prints out ignored header fields. See also **print** and **ignore.**
Reply	(R) Reply to originator. Does not reply to other recipients of the original message.
Respond	(Res) A synonym for **Reply**
Type	(T) Identical to the **Print** command.
Unread	(U) Identical to the **unread** command.
alias	(a) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates an new or changes an on old alias.
alternates	(alt) The **alternates** command is useful if you have accounts on several machines. It can be used to inform *Mail* that the listed addresses are really you. When you **reply** to messages, *Mail* will not send a copy of the message to any of the addresses listed on the *alternates* list. If the **alternates** command is given with no argument, the current set of alternate names is displayed.
chdir	(cd) Changes the user's working directory to that specified, if given. If no directory is given, then changes to the user's login directory.
copy	(c) The **copy** command does the same thing that **save** does, except that it does not mark the messages it is used on for deletion when you quit.
delete	(d) Takes a list of messages as argument and marks them all as deleted. Deleted messages will not be saved in *mbox*, nor will they be available for most other commands.
discard	(di) A synonym for **ignore.**
dp	(also **dt**) Deletes the current message and prints the next message. If there is no next message, *Mail* says "at EOF."
echo	(ec) Takes a string and echos it to standard output.

edit (e) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.

exit (ex or x) Effects an immediate return to the Shell without modifying the user's system mailbox, his *mbox* file, or his edit file in **–f**.

file (fi) The same as **folder**.

folder (fo) The **folder** command switches to a new mail file or folder. With no arguments, it tells you which file you are currently reading. If you give it an argument, it will write out changes (such as deletions) you have made in the current file and read in the new file. Some special conventions are recognized for the name. # means the previous file, % means your system mailbox, %user means user's system mailbox, & means your ~/mbox file, and +folder means a file in your folder directory.

folders List the names of the folders in your folder directory.

from (f) Takes a list of messages and prints their message headers.

group (g) A synonym for alias.

headers (h) Lists the current group of headers. Headers are grouped and displayed by windowfuls (as many headers as will fit in the window). If there are more headers than will fit in the current window, the **z** command can be used to scroll through multiple header groups.

help A synonym for ?

hold (ho, also **preserve**) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in *mbox*. Does not override the **delete** command.

ignore (ig) Add the list of header fields named to the *ignored list*. Header fields in the ignore list are not printed on your terminal when you print a message. This command is very handy for suppression of certain machine-generated header fields. The **Type** and **Print** commands can be used to print a message in its entirety, including ignored fields. If **ignore** is executed with no arguments, it lists the current set of ignored fields.

list	(**li**) Prints the list of all *Mail* commands.
mail	(**m**) Takes as argument login names and distribution group names and sends mail to those people.
mbox	(**mb**) Indicate that a list of messages be sent to *mbox* in your home directory when you quit. This is the default action for messages if you do *not* have the *hold* option set.
more	(**mo**) Like **print** but invokes your pager.
new	Identical to the **unread** command
next	(**n** like + or CR) Goes to the next message in sequence and types it. With an argument list, types the next matching message.
page	(**pa**) A synonym for **more.**
preserve	(**pre**) A synonym for **hold**.
print	(**p**) Takes a message list and types out each message on the user's terminal.
quit	(**q**) Terminates the session, saving all undeleted, unsaved messages in the user's *mbox* file in his login directory, preserving all messages marked with **hold** or **preserve** or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message "You have new mail" is given. If given while editing a mailbox file with the **–f** flag, then the edit file is rewritten. A return to the Shell is effected, unless the rewrite of edit file fails, in which case the user can escape with the **exit** command.
reply	(**r**) A synonym for **Reply**.
replyall	(**ra** or **RA**) Takes a message list and sends mail to the sender and all recipients of the specified message. The default message must not be deleted.
respond	(**res**) A synonym for **Reply**.
save	(**s**) Takes a message list and a filename and appends each message in turn to the end of the file. The filename in quotes, followed by the line count and character count is echoed on the user's terminal. If filename does not already exist it will be created. If filename begins with a "|" or "!" then it will be interpreted as a shell command and the contents of the messages passed to it on standard input.

set	(se) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form ‘‘option=value’’ or ‘‘option.’’
shell	(sh) Invokes an interactive version of the shell.
size	(si) Takes a message list and prints out the size in characters of each message.
source	(so) The **source** command reads *Mail* commands from a file.
top	(to) Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to five.
touch	(tou) Takes a message list and causes each message therein to be saved in either the user’s system mailbox or in *mbox* according to the state of the *hold* option.
type	(t) A synonym for **print**.
undelete	(u) Takes a message list and marks each one as *not* being deleted.
unread	(unr or U) Takes a message list and marks each message as *not* having been read. Also see the **Unread** command.
unset	Takes a list of option names and discards their remembered values; the inverse of set.
version	(ve) Prints the version number of Mail that you are using.
visual	(v) Takes a message list and invokes the display editor on each message.
write	(w) Like save except that the message header and the blank line after the message body are not appended to the file. Only the message body of each message is appended to the file.
xit	(x) A synonym for exit.
z	*Mail* presents message headers in windowfuls as described under the **headers** command. You can move *Mail*’s attention forward to the next window with the z command. Also, you can move to the previous window by using z–.

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name ‘‘tilde escape’’ is somewhat of a misnomer since the actual escape character can be set by the option **escape.**

~!command	Execute the indicated shell command, then return to the message.
~:command	Execute the indicated *Mail* command, then return to the message.
~?	Display the tilde escape help file.
~b name ...	Add the given names to the list of blind carbon copy recipients.
~c name ...	Add the given names to the list of carbon copy recipients.
~cm string	Cause the named string to become the current comments field.
~d	Read the file "dead.letter" from your home directory into the message.
~e	Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.
~en string	Cause the named string to become the current encrypted field.
~f messages	Read the named messages into the message being sent. If no messages are specified, read in the current message.
~h	Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.
~irt string	Add the named string to the in-reply-to list.
~k string	Add the named string to the keywords list.
~m messages	Read the named messages into the message being sent shifted right one tab. Note that if the *mprefix* option is set (see below), the tab will be replaced with the specified string. If no messages are specified, read the current message.
~p	Print out the message collected so far, prefaced by the message header fields.
~q	Abort the message being sent, copying the message to "dead.letter" in your home directory if save is set.
~r filename	Read the named file into the message.

~**rf** string	Add the named string to the references list.
~**rr**	Cause a return-receipt-to field specifying your user name to be added to the message header. Works like a toggle switch. A successive ~**rr** will remove the return-receipt-to field, a third ~**rr** will add it back, and so on. If this header field is present when the message is sent, and if the intervening mail delivery system supports return receipts, a return receipt will be sent to your mailbox when the message is successfully delivered to each of the specified recipients.
~**rt** name ...	Add the given names to the reply-to list.
~**s** string	Cause the named string to become the current subject field.
~**t** name ...	Add the given names to the direct recipient list.
~**v**	Invoke an alternate editor (defined by the VISUAL option) on the message collected so far. Usually, the alternate editor will be a screen editor. After you quit the editor, you may resume appending text to the end of your message.
~**w** filename	Write the message onto the named file.
~|command	Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command *fmt*(1) is often used as *command* to rejustify the message.
~~string	Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

Options are controlled via the **set** and **unset** commands. Options may be either binary, in which case it is only significant to see whether they are set or not, or string, in which case the actual value is of interest. The binary options include the following:

append	Causes messages saved in *mbox* to be appended to the end rather than prepended. (This is set in /usr/lib/Mail.rc on version 7 systems.)
ask	Causes *Mail* to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field will be sent.
askcc	Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint	Causes the **delete** command to behave like **dp** – thus, after deleting a message, the next one will be typed automatically.
crt	Causes your pager to be invoked if a message to be printed is longer than the current window. This option can also be used as a valued option (see below).
debug	Setting the binary option *debug* is the same as specifying **–d** on the command line and causes *Mail* to output all sorts of information useful for debugging *Mail*.
dot	The binary option *dot* causes *Mail* to interpret a period alone on a line as the terminator of a message you are sending.
hold	This option is used to hold messages in the system mailbox (instead of *mbox*) by default.
ignore	Causes interrupt signals from your terminal to be ignored and echoed as @'s.
ignoreeof	An option related to *dot* is *ignoreeof* which makes *Mail* refuse to accept a control-D as the end of a message. *Ignoreeof* also applies to *Mail* command mode.
metoo	Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.
nosave	Normally, when you abort a message with two RUBOUT, *Mail* copies the partial letter to the file "dead.letter" in your home directory. Setting the binary option *nosave* prevents this.
quiet	Suppresses the printing of the version when first invoked.
showto	Causes *Mail* to display the recipient instead of sender when displaying headers of messages for which you were the sender (your user name appears in the From: header field). This is useful when using *Mail* to browse or edit a file of saved outgoing mail such as is created when the *record* option is defined (see below).
verbose	Setting the option *verbose* is the same as using the –v flag on the command line. When mail runs in verbose mode, the actual delivery of messages is displayed on he users terminal.

The following options have string values:

EDITOR	Pathname of the text editor to use in the **edit** command and ~e escape. If not defined, then a default editor is used.
PAGER	Pathname of your pager to use in the More or more commands, or if the *crt* option is selected. If not defined, than *more* is the default.
SHELL	Pathname of the shell to use in the **!** command and the ~! escape. A default shell is used if this option is not defined.
VISUAL	Pathname of the text editor to use in the **visual** command and ~v escape.
crt	If *crt* is used as a valued option (see above for a description of how *crt* is used as a binary option), it is used as a threshold to determine how long a message must be before the user's pager is used to read it. In effect, this allows the user to override the current window size which would be used in the case of the binary *crt* option.
escape	If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.
folder	The name of the directory to use for storing folders of messages. If this name begins with a '/', *Mail* considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.
mprefix	If defined, gives the string which will be prepended to each inserted line when using the ~m command to insert text from a previous message into the current message being composed. If not defined a tab will be the default.
record	If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so saved.
replyto	If defined, gives the address which should be entered into the Reply-To header field for all outgoing mail.
toplines	If defined, gives the number of lines of a message to be printed out with the top command; normally, the first five lines are printed.

FILES

/usr/mail/*	Post office.
/usr/mail/*user*	System mailbox for *user*.
/usr/mail/*user*.lock	Lock for *user*'s mailbox.
/usr/mail/*user*.rolock	Read-only lock for *user*'s mailbox. Used to prevent file contention between
multiple	Mail instances.
~/mbox	Your old mail.
~/.mailrc	File giving initial mail commands.
~/.lsignature	File to append to outgoing local mail.
~/.rsignature	File to append to outgoing remote mail.
~/.signature	File to append to outgoing mail if .lsignature and .rsignature do not exist.
.	
/tmp/R#	Temporary for editor escape.
/usr/lib/Mail.help*	Help files.
/usr/lib/Mail.rc	System initialization file.
/usr/sbin/Mail	Mail program.
Message*	Temporary for editing messages.

SEE ALSO

mail_att(1)

BUGS

There are many flags that are not documented here. Most are not useful to the general user.

AUTHOR

Kurt Shoens

NAME

mailbox – mail notification

SYNOPSIS

mailbox [**–clbd**] [**–u** user] [**–s** size]

[**–L** latitude] [**–M** longitude] [**–S** label] [**–m** mail-program]

[**–f** mail-file] [**–C** envelope-counter] [**–B** beeper]

DESCRIPTION

mailbox watches your mailbox, and notifies you when you have mail. It displays a small picture of a mailbox to perform notification. The flag on the mailbox rises whenever new mail messages appear in your mailbox.

You may open the mailbox and read your mail by clicking on the *mailbox* window with the left mouse button. If one of the shift keys is held down, or if you have no mail, the mailbox door will simply open without running the mail reading program. Once you have finished reading your mail, the mailbox flag will go down and the door will close.

The number of envelopes in the mailbox corresponds to the number of messages in your mailbox. Manila envelopes represent five messages each. Normal letter-sized envelopes represent one envelope. *mailbox* knows how to count messages written by Berkeley and AT&T *mail*, and by *mh.*

The MAIL environment variable is used to determine where to watch for mail. If this variable does not exist, the USER environment variable is appended to the string "/usr/mail", and this file is used.

OPTIONS

–c Run in colormap mode. Ordinarily, the decision to display in colormap mode is made on the basis of available graphics hardware.

–l Run in lighted, RGB mode. This is currently a no-op.

–b Don't beep when mail arrives.

–d Print out longitude and latitude on startup.

–u user
Watch the mail of a user other than yourself.

–m <mail program>
Execute the specified program when requested to. The argument following the option must be a single string. This is generally accomplished by putting double quotes around the string. If an empty string is passed, *mailbox* will simply open and close the mailbox door when the window is clicked on.

The program string is executed by *system*(3), so environment variables and other shell substitutions can be used in this string.

Note that the program which is started by *mailbox* probably should not be a program which reads from standard input and output, since it will then try to run in the shell window which started it. If such a behavior is desired, the **−m** option should start a *wsh*(1) window to contain the program. The default **−m** command used by *mailbox* is

```
wsh -C7,97,230,95 -s40x80 -n Mail -Z1 -c /usr/sbin/Mail
```

Notice that the **−Z1** flag is necessary to make *wsh* run in the foreground.

−s size
: Set the size of the window, in points. The default size is 96 points, or 1.33 inches. The *mailbox* window is always square.

−L latitude
: Specify the latitude where the mailbox is located. This is used to determine the direction of the sun, and the times of sunset and sunrise. Latitudes south of the equator should be specified as a negative number. If no latitude is specified, 37 degrees 30' N is assumed. The latitude may also be specified through the environment variable LATITUDE, which will override the default latitude, but will be overridden by command-line options.

−M longitude
: Specify the longitude where the mailbox is located. This is used to determine the direction of the sun, and the times of sunset and sunrise. The default longitude is 122 degrees west. Longitudes are expressed as degrees west of Greenwich, England. Longitudes east of Greenwich should be specified as negative numbers. The longitude may also be specified through the environment variable LONGITUDE, which operates in the same way as LATITUDE.

−S label
: Put a small sign on top of the mailbox with the specified label printed on it. Unfortunately, the sign is hard to read if the window is small, due to resolution limits. If this bothers you, make the mailbox bigger with the −s option.

−f file Specify that *mailbox* watch a file other than your system mailbox, as though it were a mailbox. Note that *mailbox* only puts up the flag if it determines that actual mail messages have been added to the file, by counting the number of messages. However, the −C option can be used to change the way that *mailbox* counts messages.

–C envelope-counter

mailbox usually counts the number of envelopes in your mailbox every time that it notices that the mailbox has been touched. If the number of envelopes is greater than the last time it checked, it raises the flag. However, you can change *mailbox*'s concept of how to count envelopes by specifying an "envelope-counting" program.

The name of the mail file being watched will be appended to the specified string, and the result will be passed to *system*(3). The job of the counting program is to call *exit*(2) with an argument which is equal to the number of envelopes. If the counting program exits with a negative status, or if it is stopped by a signal, the number of envelopes will be assumed to be zero.

This option can be used to turn *mailbox* into a more general file-watching program, in conjunction with the –**f** and –**m** options.

–B beeper

mailbox usually beeps whenever it determines that new mail has arrived. You can override this action by providing a "beeper" program. Typical beeping programs might (on a Personal Iris) send data to /dev/audio. The beeping program is a string which will be passed to *system*(3).

FILES

$MAIL, /usr/mail/$USER

AUTHOR

Andrew Myers

SEE ALSO

mail_att(1), mail_bsd(1) in the *IRIX User's Reference Manual*.
chkconfig(1M), su(1M) in the *IRIX System Administrator's Reference Manual*.

NAME

make — maintain, update, and regenerate groups of programs

SYNOPSIS

make [**−f** makefile] [**−p**] [**−i**] [**−k**] [**−s**] [**−r**] [**−n**] [**−b**] [**−e**] [**−t**] [**−q**] [**−u**] [names]

DESCRIPTION

The *make* command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

−f *makefile*	Description filename. *makefile* is assumed to be the name of a description file.
−p	Print out the complete set of macro definitions and target descriptions.
−i	Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.
−k	Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.
−s	Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.
−r	Do not use the built-in rules.
−n	No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
−b	Compatibility mode for old makefiles.
−e	Environment variables override assignments within makefiles.
−t	Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
−q	Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.
−u	Unconditional. Build all targets regardless of whether they are up-to-date or not.
.DEFAULT	If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.

.PRECIOUS	Dependents of this target will not be removed when quit or interrupt are hit.
.SILENT	Same effect as the **–s** option.
.IGNORE	Same effect as the **–i** option.

make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **–f** option is present, **makefile**, **Makefile**, and the Source Code Control System (SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **–**, the standard input is taken. More than one **–f** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target (unless the **–u** option is present). All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date. To determine whether a dependency is out-of-date with respect to a target, the last modification times (as returned from *stat*(2)) are compared. If the macro **VPATH** is set and points to a directory, then if a dependency is not found in the current directory it is searched for the specified alternate directory. Targets are treated similarly. Note that **VPATH** does not affect any of the internal macros. Thus,

```
VPATH=object.d

foo.o:
        touch $@
```

would check for *foo.o* in both the current directory and in *object.d*, but if *foo.o* didn't exist in either place would create one in the current directory. Explicit dependencies of a target may contain the **VPATH** prefix. Thus in,

```
VPATH=object.d

foo.o:foo.q
        touch $@

$(VPATH)/foo.o:inc.h
```

the list of prerequisites for *foo.o* would contain both *foo.q* and *inc.h*.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o –o pgm
a.o: incl.h a.c
        cc –c a.c
b.o: incl.h b.c
        cc –c b.c
```

Command lines are executed one at a time, each by its own shell. The SHELL environment variable can be used to specify which shell *make* should use to execute commands. The default is */bin/sh*. The first one or two characters in a command can be the following: –, @, –@, or @–. If @ is present, printing of the command is suppressed. If – is present, *make* ignores an error. A line is printed when it is executed unless the –s option is present, or the entry .SILENT: is in *makefile*, or unless the initial character sequence contains an @. The **–n** option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the MAKEFLAGS macro under *Environment*). The **–t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **–i** option is present, or the entry .IGNORE: appears in *makefile*, or the initial character sequence of the command contains –, the error is ignored. If the **–k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **–b** option allows old makefiles (those written for the old version of *make*) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name .PRECIOUS.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **–e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **–f** and **–p**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **–n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make –n** recursively on a whole software system to see what would have been executed. This is because the **–n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string *include* or *sinclude* appears at the beginning of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros. For *include* it is a fatal error if the file is not readable, for *sinclude* a non-readable file is silently ignored.

Alternate make

make understands a convention similar to the alternate interpreter feature of *exec(2)*. If the first line of the *makefile* starts with a "#!alternate_make", then *make* will attempt to *exec* the alternate make with the same environment and arguments that *make* itself was invoked with. Additional arguments may be supplied on the "#!" line - these are placed ahead of all the command line arguments given to the original invocation of *make*. If a new makefile specification is given using the **-f** flag, any original **-f** options given on the command line are ignored. If the alternate make cannot be found in the user's **PATH** or make finds that it would be re-invoking itself, then make silently ignores the line and continues to execute the remainder of the makefile. The **-d** flag will display information as to whether the alternate make was successfully invoked. As a special case to support compatibility with a makefile used as a shell script (with a "#!/bin/make -f"), a lone **f** flag is ignored.

Macros

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1* [:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all occurrences of *subst1* at the end of a word in the value of the named macro are replaced by *subst2*. Words (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. As a special case, if *subst1* is empty, then *subst2* will be appended to each string found in *string1*. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are six internally maintained macros that are useful for writing rules for building targets.

$* The macro $* stands for the filename part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@ The $@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$$@ The $$@ macro stands for the full target name of the current target (which is $@). It has meaning only on the dependency line in a makefile. Thus, in the following:

```
cat dd: $$@.c
```

the dependency is translated at execution time first to the string **cat.c**, then to the string **dd.c**.

$< The $< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the $< macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc −c −O $*.c
```

or:

```
.c.o:
        cc −c −O $<
```

$? The $? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The $% macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

These macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .f .f~ .sh .sh~ .p .p~
.c.o .c.a .c~.o .c~.c .c~.a .c.out .c.u .c~.u
.c.b .c~.b .c.s .c~.s .c.i .c~.i .c.fc .fc.o
.f.o .f.a .f~.o .f~.f .f~.a .f.out .f.u .f~.u
.f.b .f~.b
.p.o .p.a .p~.o .p~.p .p~.a .p.out .p.u .p~.u
.p.b .p~.b
.h~.h .s.o .s~.o .s~.s .s~.a .s.out .sh~.sh .u.o
.l.o .l.c .l~.o .l~.l .l~.c .l.out
.y.o .y.c .y~.o .y~.y .y~.c .y.out
.e.o .e~.o .e.out .r.o .r~.o l.r.out
.o.out .y~.y .y~.c .y.out
```

These internal rules for *make* may be printed with the following command:

make −fp − </dev/null

Inference rules may be redefined in a *makefile* or may be completely ignored by giving the **-i** option.

A tilde in the above rules refers to an SCCS file [see *sccsfile*(4)]. Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x*.c. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .out .o .fc .u .c .c˜ .p .p˜ .f .f˜ .e .e˜ .r .r˜ .y .y˜ .l .l˜ .s .s˜ .sh .sh˜ .h .h˜ .i˜

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o –o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library that contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form *.XX*.a where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        $(AR) $(ARFLAGS) lib $?
        rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object filenames (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule by using the semicolon **;**, which would have created each object file, one-by-one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile
/bin/sh

SEE ALSO

cc(1), lex(1), yacc(1), exec(2), printf(3S), sccsfile(4).
cd(1), sh(1) in the *User's Reference Manual*.
Programmer's Guide.

NOTES

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

BUGS

Filenames with the characters = : @ will not work. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** does not work. Named pipes are not handled well. Path name compression is not done, so that *foodir* and *foodir/* will be treated as different targets.

Two additional non-standard macros **DPATH** and **IPATH** have special meaning so use of these names is discouraged. Support for these two will dropped in a future release.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

makekey is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO

ed(1), crypt(1), vi(1).
passwd(4) in the *Programmer's Reference Manual.*

NAME

makemap – make the default color map

SYNOPSIS

makemap

DESCRIPTION

When a graphics program is running in *cmode*, the frame buffer of the IRIS-4D contains values which are translated into RGB values by a color map. The default map is created by *makemap. makemap* maps the first 256 colors the same way as 4Sight. In the 4Sight map, the lowest eight colors are mapped to the eight standard colors of the Graphics Library (black, red, green, yellow, blue, magenta, cyan and white). The next 23 colors (8 to 30) are mapped as a black to white gray ramp. The remaining 225 colors (31 to 255) are mapped as a 5*9*5 color cube.

If there are more than eight planes, *makemap* maps colors 256 to 511 to the ordered color map used by some of the image processing tools in the 4DGifts package and colors 512 to 767 to a 256 level black to white gray ramp.

SEE ALSO

loadmap(1G), savemap(1G), showmap(1G),

AUTHOR

Mark Callow

NAME

man – print entries from the on-line reference manuals; find manual entries by keyword

SYNOPSIS

man [**–dwt**] [**–M** *path*] [**–T** *macropackage*] [*section*] *title* ...
man [**–M** *path*] **–k** *keyword* ...
man [**–M** *path*] **–f** *filename* ...

DESCRIPTION

man locates and prints the *title*ed entries from the on-line reference manuals. *man* also prints summaries of manual entries selected by *keyword* or by associated *filename*.

If a *section* is given, only that particular section is searched for the specified *title*. If no *section* is given, all sections of the on-line reference manuals are scached and all occurrences of *title* are printed.

Manual entries are retrieved in the following order: local additions first followed by the standard manual entries. The local additions are searched for in */usr/catman/local*. These locally added manual entries may be preformatted "cat" manual entries in */usr/catman/local/cat[1-8lnop]* or may be unformatted *nroff*(1) source manual entries in */usr/catman/local/man[1-8lnop]*. The unformatted manual entries will be processed by *neqn*(1), *tbl*(1), *nroff*(1), and *col*(1). (The *Documentor's Work Bench* software option is required to process unformatted manual pages.) The "cat" manual entries may be compressed to save disk space using *pack*(1) or *compress*(1); *man* will automatically uncompress compressed "cat" manual entries using *pcat*(1) and *zcat*(1) respectively.

After the local additions are searched, the standard pre-formatted manual entries in */usr/catman/[agpu]_man* are searched.

After searching */usr/catman*, *man* will search */usr/man* for manual pages.

The user may override the default root directories for manual entries, */usr/catman* and */usr/man*, with the environment variable **MANPATH** or with the command-line options **–M** and **–d**. (See discussion below.)

IRIX is derived from four main sources: AT&T, Berkeley, MIPS Computer Systems, and Sun Microsystems. Because development at these sources is more or less independent, in several cases programs have been given the same name but have vastly different functionality. The manual entries for such programs have been distinguished by giving them suffixes: **_att**, **_bsd**, **_mips**, or **_sun**. You do not need to give the suffixes. If *man* is given an unsuffixed title *title* for which suffixed entries exist, it will display all of them.

Searches for *titles*, *keywords*, and *filenames* are case-insensitive. For example, the manual entry *RGBcolor*(3G) can be gotten by the command-line:

```
man rgbcolor
```

Also, *titles*, *keywords*, and *filenames* may contain regular expression meta-characters *a la grep*(1) and *regex*(3X). For example, the summaries of manual entries pertaining to RGB writemasks could be searched by the command-line:

```
man –k 'rgb.*mask'
```

The regular expression handling is implemented using *regex*(3X).

OPTIONS

–M *path* Use *path* as the search path for manual entries. *path* is a colon-separated list of directories where manual subdirectories may be found. The default path is */usr/catman:/usr/man*. **–M** is useful for searching locations other than the standard manual directories for manual entries. These locations could be personal manual page trees or NFS mounted BSD style manual page trees from another system. For example, the standard manual directories could be augmented with personal manual pages by specifying the path:

```
/usr/catman:/usr/man:$HOME/man
```

–M must be given before **–k** and **–f**. **–M** will override the environment variable **MANPATH**. **–M** and **–d** are mutually exclusive.

–d Search only the current directory for the given *title*. **–d** will override the environment variable **MANPATH**. **–d** and **–M** are mutually exclusive.

–w Print only the *pathname* of each entry matching the given *title*. The actual matching entry will not be printed, only its path is given.

–t Typeset each *title*d manual entry and send the result to the printer. In the case of the preformatted "cat" manual pages which come standard with **IRIX**, the entry is unpacked using *pcat*(1) and then sent to the default printer using *lp*(1). If, however, the manual entry is a locally added, unformatted *nroff*(1) source, the entry will be formatted using *psroff*(1) and sent to the printer.

–T *macropackage*
: The given *nroff*(1) macro package will be used for formatting unformatted manual entries. By default, */usr/lib/tmac/tmac.an* is used.

–k *keyword* Print the manual entry summaries which contain the given *keywords*. The summaries are gotten from the *whatis*(4) database. (See also *apropos*(1).)

–f *filename* Print the manual entry summaries which might pertain to the given *filenames*. Any leading pathname components are stripped from the *filename* before the *filename* is matched against the summaries. The summaries are gotten from the *whatis*(4) database. (See also *whatis*(1).)

ENVIRONMENT

MANPATH If set, **MANPATH** overrides the default manual entry search path */usr/catman:/usr/man*. **MANPATH** is a colon-separated list of directories where manual subdirectories may be found. (See the discussion of –M.) **–M** and **–d** will override **MANPATH**.

PAGER and **MANPAGER**
: If set, **PAGER** and **MANPAGER** specify a program for interactively displaying the output from *man*. **MANPAGER** will override **PAGER** so a program other than the user's standard paging program may be used for displaying *man* output. If neither **PAGER** or **MANPAGER** are set, the command "ul | more -s -f -k" is used.

FILES

/usr/catman	root directory of on-line reference manual entry tree
/usr/catman/whatis	table of contents and keyword database
/usr/catman/u_man/cat[1,6]/*	*IRIS-4D User's Reference Manual*
/usr/catman/a_man/cat[1,7]/*	*IRIS-4D System Administrator's Reference Manual*
/usr/catman/p_man/cat[2-5]/*	*IRIS-4D System Programmer's Reference Manual*
/usr/catman/g_man/cat3/*	*Graphics Library User's Guide, Volume II, Reference Guide*
/usr/catman/local/cat[1-8lnop]/*	local pre-formatted manual entries
/usr/catman/local/man[1-8lnop]/*	local unformatted *nroff*(1) source manual entries

/usr/man/*	additional manual pages
/usr/lib/tmac/tmac.an	default macro package used for formatting manual entries

SEE ALSO

apropos(1), col(1), compress(1), eqn(1), grep(1), lp(1), makewhatis(1M), more(1), neqn(1), nroff(1), pack(1), pcat(1), psroff(1), tbl(1), whatis(1), zcat(1), regex(3X), environ(5), man(5), term(5).

CAVEATS

The */usr/catman* directories have all been processed by *nroff*(1). AT&T does not allow Silicon Graphics, Inc. to ship the *nroff*(1) sources for manual pages.

The *Documentor's Work Bench* software option is required to process locally added, unformatted *nroff (1)* source manual pages.

man will not locate manual pages in directories with names containing a period (.).

NAME

mapimg – translates a screen image into an RGB image

SYNOPSIS

mapimg image.sc image.rgb temp.map

DESCRIPTION

mapimg translates a screen image into a full RGB image using the given color map. The color map is usually generated by *savemap*.

NAME

maze – an automated maze program... [demo][X11]

SYNTAX

maze [–S] [–r] [–g *geometry*] [–d *display*]

DESCRIPTION

The *maze* program creates a "random" maze and then solves it with graphical feedback.

Command Options

–S Full screen window option...

–r Reverse video option...

–g *geometry*
Specifies the window geometry to be used...

–d *display*
Specifies the display to be used...

The following lists the current functionality of various mouse button clicks;

LeftButton
Clears the window and restarts maze...

MiddleButton
Toggles the maze program, first click -> *stop*, second click -> *continue...*

RightButton
Kills maze...

LIMITATIONS

No color support...
Expose events force a restart of maze...
Currently, mouse actions are based on "raw" values [Button1, Button2 and Button3] from the ButtonPress event...
[doesn't use pointer mapping]

COPYRIGHT

AUTHOR(s)

Richard Hess [X11 extensions] {...}!uunet!cimshop!rhess
Consilium, Mountain View, CA
Dave Lemke [X11 version] lemke@sun.COM
Sun MicroSystems, Mountain View, CA
Martin Weiss [SunView version]
Sun MicroSystems, Mountain View, CA

NAME

merge — three-way file merge

SYNOPSIS

merge [**-p**] file1 file2 file3

DESCRIPTION

Merge incorporates all changes that lead from *file2* to *file3* into *file1*. The result goes to standard output if **-p** is present and into *file1* otherwise. *Merge* is useful for combining separate changes to an original. Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then *merge* combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. *Merge* prints the number of overlaps that occurred and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result and delete one of the alternatives.

DIAGNOSTICS

If the *diff3* program which *merge* uses fails, *merge* prints the error message from *diff3* and exits with a non-negative status.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 82/11/25.
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

diff3(1), diff(1), rcsmerge(1), co(1).

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [**–n**] [**–y**]

DESCRIPTION

mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdepend – compute header file dependencies

SYNOPSIS

mkdepend [**–c** *compiler*] [**–e** *sedprog*] [**–f** *force*] [**–i**] [**–p** *count*] [**–r**] [**–s** *sentinel*] *depfile* [*file* ...]

DESCRIPTION

Mkdepend infers *make*(1) dependencies from source containing C #include directives. It invokes *cc*(1) with the **–M** option to compile dependencies given a list of source *files*, and edits the generated dependency information into *depfile*, which may be a makefile or a make include file.

The **–c** option substitutes *compiler*, which may be an elaborate, quoted invocation of a compiler, for the default **cc –M**. This option is useful in an environment where the **–D** and **–I** options of *cc* are used to govern header file inclusion.

The **–e** option passes an editing program to *sed*(1), which is applied to raw dependency information of the following form:

```
object: source
```

Thus one may substitute pathname prefixes with envariable parameters, or rewrite the conventional **.o** object suffix.

The **–f** option causes *mkdepend* to add a dependent named *force* to each target file's dependency list. Using **–f '$(FORCE)'** and setting **FORCE=FORCE** in a make's environment, one may rebuild certain objects without first removing them. This example assumes that no file named **FORCE** exists in any of *make*'s current working directories.

Normally, existing dependencies are deleted from the makefile. The **–i** option causes *mkdepend* to preserve old dependencies, replacing only those involving targets based on the *file* arguments. The following rule incrementally updates a dependency file named **Makedepend**:

```
Makedepend: $(CFILES)
        mkdepend –c "$(CC) $(CFLAGS) -M" –i $@ $?
```

The **–r** option causes *mkdepend* to read raw dependencies from its *file* arguments, or from standard input if no *file* arguments are given. The **–p** option is like **–r**, but also collapses dependencies of targets in subdirectories where possible. The *count* argument specifies the number of subdirectories. For example,

```
subdir1/object: source
subdir2/object: source
subdir3/object: source
```

would be rewritten by **mkdepend –p 3** as

```
object: source
```

–p is useful in conjunction with *make*'s **VPATH** variable.

The **–c**, **–p**, and **–r** options override one another.

The –s option inserts its *sentinel* argument in the comment lines which delimit dependencies in *depfile*. This option is useful for maintaining multiple sets of dependencies in a single file.

SEE ALSO

cc(1), cpp(1), make(1), sed(1)

AUTHOR

Brendan Eich

NAME

mkdir – make directories

SYNOPSIS

mkdir [**–m** mode] [**–p**] dirname ...

DESCRIPTION

mkdir creates the named directories in mode 777 (possibly altered by *umask*(1)).

Standard entries in a directory (e.g., the files ., for the directory itself, and .., for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

–m This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).

–p With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

sh(1), rm(1), umask(1).
intro(2), mkdir(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

mkdir returns exit code 0 if all directories given in the command line were created successfully. Otherwise, it prints an error message and returns a non-zero exit status.

NAME

mkfontdir - create fonts.dir file from directory of font files.

SYNOPSIS

mkfontdir [directory-names]

DESCRIPTION

Mkfontdir For each directory argument, mkfontdir reads all of the font files in the directory searching for properties named "FONT", or (failing that) the name of the file stripped of its suffix. These are used as font names, which are written out to the file "fonts.dir" in the directory along with the name of the font file.

The kinds of font files read by mkfontdir depends on configuration parameters, but typically include SNF (suffix ".snf"), compressed SNF (suffix ".snf.Z"), BDF (suffix ".bdf"), and compressed BDF (suffix ".bdf.Z"). If a font exists in multiple formats, the most efficient format will be used.

FONT NAME ALIASES

The file "fonts.alias" which can be put in any directory of the font-path is used to map new names to existing fonts, and should be edited by hand. The format is straight forward enough, two white-space separated columns, the first containing aliases and the second containing font-name patterns.

When a font alias is used, the name it references is search for in the normal manner, looking through each font directory in turn. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white-space in either name, simply enclose them in double-quote marks, to embed double-quote marks (or any other character), precede them with back-slash:

```
"magic-alias with spaces"  "\"font\name\" with quotes"
regular-alias                        fixed
```

If the string "FILE_NAMES_ALIASES" stands alone on a line, each file-name in the directory (stripped of it's .snf suffix) will be used as an alias for that font.

USAGE

Xserver(1) looks for both "fonts.dir" and "fonts.alias" in each directory in the font path each time it is set (see xset(1)).

SEE ALSO

X(1), Xserver(1), xset(1)

NAME

mkf2c – generate FORTRAN-C interface routines

SYNOPSIS

mkf2c [options] [cprog.fc [cprog.s]]

DESCRIPTION

mkf2c is used to generate assembly-language routines to provide greater flexibility when calling a C function from a FORTRAN routine.

Mkf2c accepts as input a set of C functions, and produces an assembly-language interface routine in the output file. If the input and output files are not specified, *mkf2c* reads from *stdin* and writes to *stdout.* The input may be a copy of the actual C file being interfaced, perhaps filtered by the program *extcentry*(1). The output of *mkf2c* is an assembly-language (*.s*) file that must be assembled with *as*(1), and loaded with the FORTRAN and C routines that are to be interfaced.

Mkf2c uses the parameter declarations in the C function headers to transform each parameter of the calling language to that of the receiving language. The standard basic C types attached to the parameters are used to determine the object each parameter represents – i.e., whether it is a value or pointer, its size, whether it is unsigned, etc. Only the opening and closing brace of the function body must be present. Information in the body of the function is ignored. *mkf2c* expects its input to consist solely of the functions it is to interface, comments, and lines which begin with the preprocessor control character '#'. It can match braces, enabling it to bound function bodies. It cannot, however, understand other C constructs normally occurring at the global level (typedefs, structure declarations, data declarations, function prototypes, etc.). Such unrecognized constructs must be eliminated from the input (this is the purpose of *extcentry*(1)).

Mkf2c will ignore functions of storage class *static* .

Make(1) contains default rules for generating an object file which consists of a file of C functions and their associated interface routines. If some functions in a C source file *foo.c* are to be made FORTRAN-callable with special interface routines included, those functions to be interfaced should be surrounded by the special comments /* CENTRY */ and /* ENDCENTRY */ (for *extcentry*(1)), and a dependency of *foo.o* on *foo.fc* should be inserted in the makefile. This dependency will cause *extcentry*(1) to be run on *foo.c*, producing *foo.fc*, *mkf2c* to be run on *foo.fc*, and the resultant *.s* file to be assembled and combined with the file produced by compiling *foo.c* with *cc(1).*

The options to *mkf2c*(1) are the following:

–f
: Suppress extending floats to doubles across the call. Normally, formal parameters of type *float* in the C input to *mkf2c* are dereferenced and converted to type *double* across the interface, to conform to C calling conventions. This option suppresses the conversion to *double*. If this option is selected, the receiving routine in C should have a *prototype* with the *float* parameters declared correctly.

–o *output*
: Name the output file *output*. If the output filename is not specified by a *–o filename* switch, *mkf2c* will use the second filename appearing in its argument list as the output file name. This method must be used if it is desired to generate an interface routine in a file when the input is from *stdin*.

–U
: Normally, upper case characters appearing in FORTRAN external names are mapped to lower case. This option suppresses that mapping, allowing FORTRAN external names to be of mixed case. This option should be used in conjunction with the *–U* option to *f77(1)*.

–signed,–unsigned
: Specify the *signed* attribute of single-character parameters. The setting of this option determines whether a scalar parameter of type *char* (in the C input to *mkf2c*), which corresponds to a FORTRAN argument of type *character*1*, should be sign-extended across the interface. The default setting is *unsigned*.

–l
: By default, *mkf2c* truncates FORTRAN external names to *six* characters to conform to the ANSI standard and to be backwards-compatible with the IRIS 4D Series. This switch allows the maximum number of characters in FORTRAN external names to be the same as that enforced by the FORTRAN front-end (currently 32). If this switch is *not* specified, the FORTRAN program should have the C function name truncated to *six* characters at the call.

–v
: Inhibit the generation of warning messages. As creating *wrappers* can cause confusion, *mkf2c* gives warning messages for constructs which will result in an interface which is 'unnatural' for C (i.e., in which the C side of the interface must take special precautions when accessing the parameters or naming the routines). An example of this would be passing a FORTRAN character variable as a C character array. *Mkf2c* knows that this situation requires C to use special care when manipulating the string, as it is not null-terminated, and, hence, it

generates a warning message. It is recommended that *−v* only be used by programmers experienced with the generation of *wrappers*.

EXAMPLE

In this example, a FORTRAN program wants to call a C function *AllParameters* with many parameters. The FORTRAN program is in the file *f.f* and the C function is in the file *c.c*. These are the only two files in the program. The C function header is given below:

```
/* CENTRY */
AllParameters(i,s,c,cptr,ptr1,ptr2,ar1,f,d,d1,struct1,string1,string2,u)
short s;
char c,*cptr;
int *ptr1;
char *ptr2[];
short ar1[];
float f;
double d,*d1;
struct test_s *struct1;
char string1[],string2[30];
sometype u;
{
        /*
        The C function body is ignored by mkf2c.
        */
}
/* ENDCENTRY */
```

When this function is run through *mkf2c*, a complaint will be given about not understanding the type of parameter *u*. It will be assumed to be a simple pointer. Additionally, a warning about passing the parameters *string1* and *string2* as simple pointers will be given. (These FORTRAN character variables each have an associated length which is passed as a *hidden* parameter to the C function, at the end of the parameter list. These additional parameters may be accessed by the C function by the use of the *varargs* macros. See the *FORTRAN Language Reference Manual* for more information.)

The parameter *i* will be assumed to be of type *int*, as it is by the C compiler *ccom* during compilation.

Several items are noteworthy about the parameters in this example. The parameters *i*, *s*, *c*, *f*, and *d* will be dereferenced accross the call. The parameter *f* will be extended to a *double* across the call unless the *–f* switch is given to *mkf2c*(1). The parameters *ptr1*, *ptr2*, *ar1*, *d1*, *struct1*, *string1*, and *string2* will be passed as simple pointers. The FORTRAN character*1 variable which is passed as *c* will be dereferenced and extended to a long across the call. If the *–signed* switch is specified, *c* will be sign-extended when being dereferenced. A copy of parameter *cptr* will be made and the copy null-terminated. A pointer to this copy will be passed. The C entry point will be named *AllParameters*. The FORTRAN entry point name depends on whether or not the *–U* and/or the *-l* switches have been given. The various combinations of these switches and their effect is detailed below:

Switches	FORTRAN Entry
<none>	allpar_
–l	allparameters_
–U	AllPar_
–l –U	AllParameters_

The program can be made easily using the built-in rules in *make*(1). The makefile would be as follows:

```
test:   f.o c.o
        f77 -o test f.o c.o

c.o: c.fc

clean:
        rm -f *.o test *.fc
```

In the make, the program *extcentry* will be run on *c.c* to produce *c.fc*. This program (see *extcentry*(1)) will copy to *c.fc* all text in *c.c* which is between the special comments */* CENTRY */* and */* ENDCENTRY */*. *Mkf2c* will then be run on *c.fc*, and the make variable *F2CFLAGS* will be passed to it. The C source will be compiled with *cc*(1), and the output of *mkf2c* will be assembled. These two *.o*s will then be loaded together into a single relocatable named *c.o*.

If it is desired to pass *mkf2c*(1) some flags (e.g., *–l* and *–signed*), the *make* variable *F2CFLAGS* should be set in the makefile, as

```
F2CFLAGS = -signed -l
```

SEE ALSO

extcentry(1), cc(1), FORTRAN Language Reference Manual

DIAGNOSTICS

Mkf2c is very simple-minded about diagnosing syntax errors. It can detect such things as a formal parameter having its type declared when it is not in the formal parameter list. A few such cases give intelligible error messages. The program will complain about types it does not understand. The default type assigned in such cases is *simple pointer*. *Mkf2c* will also delete characters from FORTRAN entry names which are illegal (e.g., underscores). The user will be warned in such instances. Most errors that the programs detect are indicated only by the source line number.

If *mkf2c* encounters an error which it cannot remedy, it will abort, giving the line number on which the error occurred. The resultant *.s* file will be removed, and an error exit will be taken.

Because of its limited error diagnostic ability, it is advisable to use *cc (1)* to determine whether the input syntax is correct before passing it to *mkf2c*.

AUTHOR

Greg Boyd

NAME

mkshlib – create a shared library

SYNOPSIS

mkshlib –s specfile [**–t** target] [**–h** host] [**–n**] [**–q**] [–v]

DESCRIPTION

The *mkshlib* command builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs that link with a shared library will share the library code during execution. Programs that link with a non-shared library will get their own copies of each library routine used.

The host shared library is an archive that is used to link-edit user programs with the shared library [see *ar*(4)]. A host shared library can be treated exactly like a non-shared library and should be included on compiler driver (*cc*(1), etc.) command lines in the usual way. Further, all operations that can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module that is attached to the user's process during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes that use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to *mkshlib* consists of command line options and a shared library specification file, *specfile*. The shared library specification file describes the contents of the shared library.

The *mkshlib* command invokes other tools, such as the archiver, *ar*(1), the assembler, *as*(1), and the link editor, *ld*(1). Tools are invoked through the use of *execvp*(3), which searches directories in the user's PATH. Also, suffixes to *mkshlib* are parsed in the same manner as suffixes to the compiler drivers, and invoked tools are given the suffix, where appropriate. For example, mkshlib*1.30* will invoke ld*1.30*.

The following command line options are recognized by *mkshlib*:

–s *specfile* Specifies the shared library specification file, *specfile*. This file contains the information necessary to build a shared library. Its contents include the branch table specifications for the target, the pathname in which the target should be installed, the start addresses of text and data for the target, the initialization specifications for the host, and the list of object files to be included in the shared library (see details below).

–t *target*
: Specifies the name, *target*, of the target shared library produced on the host machine. When *target* is moved to the target machine, it should be installed at the location given in the specification file (see the **#target** directive below). If the **–n** option is used, a new target shared library will not be generated.

–h *host*
: Specifies the name of the host shared library, *host.* If this options is not given, the host shared library will not be produced.

–n
: Do not generate a new target shared library. This option is useful when producing only a new host shared library. The –t option must still be supplied since a version of the target shared library is needed to build the host shared library.

–q
: Quiet warning messages. This option is useful when warning messages are expected, but not desired.

–v
: Set the verbose option. This option prints the command lines it executes as in the compiler drivers.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the following directives:

#address segname address
: Specifies the start address, *address*, of the segment *segname* for the target. This directive is used to specify the start addresses of the text and data segments. Since the headers part of the text segment of target shared libraries they are put on their own page. Actual library text is placed one page above the *address* specified in the directive. Legal *segnames* are *.text* and *.data.* See the **NOTES** section for information regarding *address* selections.

#target pathname
: Specifies the absolute pathname, *pathname*, of the target shared library on the target machine. This pathname is copied to **a.out** files and is the location where the operating system will look for the shared library when executing a file that uses it.

#branch Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines.

Branch table specification lines have the following format:

funcname <*white-space*> **position**

where *funcname* is the name of the symbol given a branch table entry and *position* specifies the position of *funcname's* branch table entry. *Position* may be a single integer integer or a range of integers of the form *position1–position2*. Each *position* must be greater than or equal to one, the same position cannot be specified more than once, and every position from one to the highest given position must be accounted for.

If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library.

Finally, only functions should be given branch table entries, and those functions must be external.

This directive can be specified only once per shared library specification file.

#objects Specifies the names of the object files constituting the target shared library. The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each filename followed by white space. This list is also used to determine the input object files for the host shared library.

This directive can be specified only once per shared library specification file.

Each file listed under the **#objects** directive *must be compiled with the –G 0 compiler option.*

#init object
: Specifies that the object file, *object*, requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

pimport *<white-space>* **import**

Pimport is a pointer to the associated imported symbol, *import*, and must be defined in the current specified object file, *object*. The initialization code generated for each such line is of the form:

pimport = &import;

where *pimport* is the absolute address of *import*.

All initializations for a particular object file must be given at once and multiple specifications of the same object file are not allowed.

#ident string
: Specifies a string, *string*, to be included in the .comment section of the target shared library. This directive can be specified only once per shared library specification file. This is ignored but allowed for compatibility.

##
: Specifies a comment. All information on a line following this directive is ignored.

All directives that may be followed by multi-line specifications are valid until the next directive of the end of the file.

FILES

TEMPDIR/*	temporary files

TEMPDIR is usually /tmp, but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*() in *tmpnam*(3S)].

SEE ALSO

ar(1), as(1), cc(1), ld(1), a.out(4), ar(4)

NOTES

To avoid conflict with system shared libraries and to conform with the addressing restrictions on the MIPS processor, user shared libraries must have the addresses of their text and data segments (i.e., the addresses specified in the **#address** directives) on 4-megabyte boundaries at the highest possible addresses below 0x0c000000. The data segment must be placed at a higher address than the text segment.

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name.

To process the error messages in the source to the message file *mkstr* keys on the string '**error("**' in the input stream. Each time it occurs, the C string starting at the " is placed in the message file followed by a new-line character and a null character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains an *lseek*(2) pointer into the file which can be used to retrieve the message, i.e.:

```
char  efilname[] = "/usr/lib/pi_strings";
int   efil = -1;

error(a1, a2, a3, a4)
{
      char buf[256];

      if (efil < 0) {
            efil = open(efilname, 0);
            if (efil < 0) {
oops:
                  perror(efilname);
                  exit(1);
            }
      }
      if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
            goto oops;
      printf(buf, a2, a3, a4);
}
```

The optional – causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr*ed program.

EXAMPLE

If the current directory has files "a.c" and "b.c", then

```
        mkstr exs x *.c
```

would create a new file "exs" which holds all the error messages extracted from the source files "a.c" and "b.c", as well as two new source files "xa.c" and "xb.c" which no longer contains the extracted error messages.

SEE ALSO

lseek(2).

BUGS

All the arguments except the name of the file to be processed are unnecessary.

AUTHORS

Bill Joy and Charles Haley.

NAME

more, page – file perusal filter for crt viewing

SYNOPSIS

more [**–cdflsun**] [+linenumber | +/pattern] [name ...]

page [**–cdflsun**] [+linenumber | +/pattern] [name ...]

DESCRIPTION

more is a filter which allows examination of a continuous text one screenful at a time on a CRT terminal. It normally pauses after each screenful, printing "--More--" at the bottom of the screen.

If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. If a space is preceded by an integer, that number of lines is printed. If the user hits **d** or control-D, 11 more lines are displayed (a "scroll").

more looks in the file */usr/lib/terminfo* to determine terminal characteristics and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the "--More--" prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

The following options are available:

–n Integer size (in lines) of the window which *more* will use instead of the default.

–c *more* will draw each page by beginning at the top of the screen and erasing each line just before it draws it. Redrawing the screen in this manner, avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of the line.

–d At the end of each screenful, causes *more* to prompt the user with the message "Hit space to continue, Rubout to abort".

–f Causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus, *more* may think that lines are longer than they actually are, and fold lines erroneously.

–l Causes *more* not to treat control-L (form feed) specially. If this option is not given, *more* will pause after any line that contains a control-L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

–s Squeeze multiple blank lines from the output, producing only one blank line. This option is especially helpful when viewing *nroff* output. This option maximizes the useful information presented on the screen.

–u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal. If the terminal can perform underlining or has a standout mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **–u** option suppresses this processing.

+linenumber
Causes *more* to start up at *linenumber*

+/pattern
Causes *more* to start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as *page* then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and *k–1* rather than *k*–2 lines are printed in each screenful, where *k* is the number of lines in the terminal display.

Once inside *more*, other sequences may be typed when *more* pauses. The sequences and their effects are as follows (*i* is an optional integer argument, defaulting to 1) :

*i*z Same as typing a space except that *i*, if present, becomes the new window size.

*i*s Skip *i* lines and print a screenful of lines

*i*f Skip *i* screenfuls and print a screenful of lines

*i*b Skip back *i* screenfuls and print a screenful of lines

i^B Skip back *i* screenfuls and print a screenful of lines

*i*n Skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

*i*p Skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

q or Q
: Exit from more.

i/*expr*
: Search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr* and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*
: Invoke a shell with *command.*

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the "--More--(xx%)" message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). *More* will stop sending output, and will display the usual "--More--" prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. Thus, what you type does not show on your terminal, except for the '/' and '!' commands.

If the standard output is not a terminal, then *more* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

```
        nroff −ms +2 doc.n | more
```

would show the *nroff* output on the terminal screen.

FILES

/usr/lib/terminfo	terminal data base
/usr/lib/more.help	help file

NAME

mpadmin – control and report processor status

SYNOPSIS

mpadmin –n

mpadmin –u[processor]

mpadmin –r[processor]

mpadmin –c[processor]

mpadmin –s

DESCRIPTION

mpadmin provides control/information of processor status.

Exactly one argument is accepted by *mpadmin* at any particular invocation. The following arguments are available:

–n Report which processors are physically configured. The numbers of the physically configured processors are printed on the standard output, one processor number per line. Processors are numbered beginning from 0.

–u[*processor*]
If no *processor* is specified, the numbers of the processors that are available to schedule unrestricted processes are printed on the standard output. Otherwise, *mpadmin* enables processor numbered *processor* to run any unrestricted processes.

–r[*processor*]
With no *processor* specified, the numbers of the processors that are restricted from running any processes (except those assigned to it via a *sysmp MP_MUSTRUN* command, a *runon(1)* command, or because of hardware necessity) are printed on the standard output. Otherwise, *mpadmin* restricts processor numbered *processor.*

–c[*processor*]
With no *processor* specified, the number of the processor that handles the operating system software clock is printed on the standard output. Otherwise, the operating system software clock handling is moved to the processor numbered *processor*. The fast clock processor(see ftimer(1)) is always the same processor as the clock processor.

A summary of the unrestricted, restricted and clock processors
is printed to the standard output.

SEE ALSO

runon(1), sysmp(2).

DIAGNOSTICS

When an argument specifies a *processor*, 0 is returned on success, -1 on failure. Otherwise, the number of processors associated with *argument* is returned.

WARNINGS

It is not be possible to restrict all processors, nor are all processors restrictable.

BUGS

Changing the clock processor may cause the system to lose a small amount of system time.

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**–t tapename**] command [count]

DESCRIPTION

mt is used to give commands to the magnetic tape drives. By default, *mt* performs the requested operation using */dev/nrtape*. Normally the operations are performed once. Some operations may be performed multiple times by specifying *count*. For all others, *count* is ignored.

To use an alternate device, the option **–t tapename** may be used.

The **tapename** field can also reference a remote tape device. A remote tape device name has the form:

*[user@]*system:/dev/???

Where *system* is the remote system, */dev/???* is the particular drive on the remote system (raw, rewinding, non-rewinding, etc.), and the optional *user* is the login name to be used on the remote system (the default is the current login name).

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified. Note that not all commands are implemented for all devices.

weof	Write *count* end-of-file marks at the current position on the tape.
fsf	Forward space *count* files.
fsr	Forward space *count* records (blocks).
bsf	Backspace *count* files.
bsr	Backspace *count* records (blocks).
rewind	Rewind the tape
feom	Forward space to end of recorded data. This allows appending new partitions to a tape that already contains data.
offline	Rewind and unload the tape from the drives heads, allowing removal of the tape. The tape is not ejected automatically, even for drives that support it.

unload Inform the drive it is OK for the tape to be removed. This is meaningful only for drives such as the 8mm drive that enable and disable the eject button under software control. Some drives will otherwise not allow tape removal if the tape hasn't been previously rewound. If the drive supports it, the tape is ejected.

erase Erase from current position to EOT. This can be very slow for some tape drives (up to 2 hours for 8mm tape drives with 2.3Gb capacity).

exist Exit with status 0 if the drive exists, otherwise non-zero; this is primarily for use in scripts.

recerron
Enable soft error reporting for drives that are very verbose about them, such as Cipher 540S; this persists until explicitly turned off. The default is **recerroff**.

recerroff
Enable soft error reporting for drives that are very verbose about them, such as Cipher 540S. This is the default behavior.

sili Suppress illegal length indicator (this occurs on tapes like the 8mm tape drive when a request is made to read fewer bytes than a block was written with, when in variable block mode). The default is to return a short count, when set, no short count will be returned, and the rest of the data in the block will be skipped. This is off by default. This persists until explicitly turned off or the tape is changed.

eili Reverses the effect of sili (returns to default).

reset Resets the tape drive and/or controller. This can sometimes cause the drive to not respond for some period of time. For SCSI tape drives, this means resetting all SCSI devices, including disk drives. This may cause some loss of data, and may cause devices to be unavailable for up to several minutes. It should only be used as a last resort, when the only other choice seems to be to reboot the system.

status Print status information about the tape unit. A tape cartridge need not be physically loaded to obtain status. For SCSI drives, the low 16 bits of drive status are printed. These bits are defined in */usr/include/sys/tpsc.h*. For

Pertec interface drives and ISI QIC-24 drives, the device status registers are printed as the field "Errors"; the meaning of these bits are defined in */usr/include/sys/xmreg.h* and */usr/include/sys/tsreg.h* respectively. The meaning of these bits may vary from release to release, and is primarily useful for reporting hardware problems to the service organization.

blksize Print the default block size to be used by tar. A tape cartridge need not be physically loaded to obtain default block size. The maximum, minimum, and current blocksizes are also reported; they may all be the same if the drive does not support variable block sizes.

retension
Retension the tape in the drive.

help Print a summary of the available options.

mt returns a 0 exit status when the operations were successful, a 1 if a command was unrecognized, and a 2 if a operation failed. *mt* without any arguments defaults to **help**.

FILES

/dev/tape	default tape device
/dev/nrtape	No rewind form of default tape device

SEE ALSO

mtio(7), rmt(1M), rmtops(3), tps(7m), ts(7m), xmt(7m)

NAME

multgrps – spawn a shell with process membership in multiple groups

SYNOPSIS

multgrps

DESCRIPTION

multgrps initializes the group-set of which the calling process (user) is a member. Via an *initgroups(3X)* call, multgrps obtains the groups (from file /etc/groups) and associates them with the process, then spawns a new shell which inherits them. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new set of group IDs. The user is always given a new shell, replacing the current shell, by *multgrps*, regardless of whether the user is a member of any other groups. In that shell the first multiple group in the list is always the group id from the user's entry in the /etc/passwd file.

Exported variables retain their values after invoking *multgrps*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than $ (default) and has not exported PS1. After an invocation of *multgrps* their PS1 will now be set to the default prompt string $. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

The multiple-group shell may be exited via "exit", which returns to the previous shell.

No group password checking is performed; since /etc/group is a system file (writeable only by superuser) it is assumed that protection against unintended group-membership is provided by those restricted file permissions.

Multiple groups is a BSD-style enhancement; *Multgrps* allows the convenient use of this feature without compromising the System V semantics. A user desiring multiple-group functionality may invoke *multgrps*, otherwise the default System V permission-checking is used.

The set of active group ids may be displayed by invoking *id(1)*. If *multgrps* has not been called then only the uid and gid (from the /etc/passwd file) will display.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), multgrps(1), id(1), getgroups(2), setgroups(2), initgroups(3X), sh(1) in the *User's Reference Manual*.
group(4), passwd(4), environ(5) in the *Programmer's Reference Manual*.

NAME

muncher – draw interesting patterns in an X window

SYNOPSIS

muncher [-option ...]

OPTIONS

–r	display in the root window
–s *seed*	seed the random number seed
–v	run in verbose mode
–q	run in quite mode

–geometry *geometry*
define the initial window geometry; see *X(1)*.

–display *display*
specify the display to use; see *X(1)*.

DESCRIPTION

Muncher draws some interesting patterns in a window.

SEE ALSO

X(1)

BUGS

There are no known bugs. There are lots of lacking features.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

NAME

nawk – pattern scanning and processing language

SYNOPSIS

nawk [–F *re*] [*parameter...*] ['*prog*'] [–**f** *progfile*] [*file...*]

DESCRIPTION

nawk is a new version of *awk* that provides capabilities unavailable in previous versions. This version will become the default version of *awk* in the next major UNIX system release.

The –**F** *re* option defines the input field separator to be the regular expression *re*.

Parameters, in the form x=... y=... may be passed to *nawk*, where x and y are *nawk* built-in variables (see list below).

nawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (′) to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the –**f** *progfile* option.

Input files are read in order; if there are no files, the standard input is read. The file name – means the standard input. Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the –**F** *re* option.) The fields are denoted **$1**, **$2**, . . . ; **$0** refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (**!**, **| |** , **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

```
expression relop expression
expression matchop regular expression
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (contains) or !~ (does not contain). A conditional is an arithmetic expression, a relational expression, the special expression

var **in** array,

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read respectively.

Regular expressions are as in *egrep* [see *grep*(1)]. In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second pattern.

A regular expression may be used to separate fields by using the –F *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default %.6g)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

An action is a sequence of statements. A statement may be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
do statement while ( conditional )
for ( expression ; conditional ; expression ) statement
for ( var in array ) statement
delete array[subscript]
break
continue
{ [ statement ] ... }
expression        # commonly variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next              # skip remaining patterns on this input line
exit [expr]       # skip the rest of the input; exit status is expr
return [expr]
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators +, —, *, /, %, and concatenation (indicated by a blank). The C operators ++, —, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The **print** statement prints its arguments on the standard output, or on a file if *>expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to the format [see *printf*(3S) in the *Programmer's Reference Manual*].

nawk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2*, *cos*, *exp*, *int*, *log*, *rand*, *sin*, *sqrt*, and *srand*. *int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* (expr) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(for, repl, in)
: behaves like *sub* (see below), except that it replaces successive occurrences of the regular expression (like the *ed* global substitute command).

index(s, t)	returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.
length(s)	returns the length of its argument taken as a string, or of the whole line if there is no argument.
match(s, re)	returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.
split(s, a, fs)	splits the string *s* into array elements *a*[*1*], *a*[2], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.
sprintf(fmt, expr, expr, . . .)	formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.
sub(for, repl, in)	substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, *nawk* substitutes in the current record ($0).
substr(s, m, n)	returns the *n*-character substring of *s* that begins at position *m*.

The input/output and general functions are:

close(filename)	closes the file or pipe named *filename*.
cmd | *getline*	pipes the output of *cmd* into *getline*; each successive call to *getline* returns the next line of output from *cmd*.
getline	sets $0 to the next input record from the current input file.
getline <file	sets $0 to the next record from *file*.
getline var	sets variable *var* instead.
getline var <file	sets *var* from the next record of *file*.
system(cmd)	executes *cmd* and returns its exit status.

All forms of *getline* return 1 for successful input, 0 for end of file, and −1 for an error.

nawk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

```
function name(args,...) { stmts }
func name(args,...) { stmts }
```

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

```
      { s += $1 }
END   { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Simulate *echo*(1):

```
BEGIN {
      for (i = 1; i < ARGC; i++)
            printf "%s", ARGV[i]
      printf "\n"
      exit
      }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: **nawk –f program n=5 input**

SEE ALSO

grep(1), sed(1).
lex(1), printf(3S) in the *Programmer's Reference Manual*.
Programmer's Guide.

BUGS

Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

netstat – show network status

SYNOPSIS

netstat [**–Aanu**] [**–f** *address_family*] [*system*] [*core*]
netstat [**–imnqrsM**] [**–f** *address_family*] [*system*] [*core*]
netstat [**–n**] [**–I** *interface*] *interval* [*system*] [*core*]
netstat [**–p** *protocol*] [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an *interval* specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces. The fourth form displays statistics about the named protocol.

The options have the following meaning:

–A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

–a With the default display, show the state of all sockets; normally sockets used by server processes are not shown.

–i Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown). When –a is also present, show all addresses (unicast and multicast) associated with each interface.

–iq Show the information for –i with the number of packets currently in the output queue, the queue size, and the number of dropped packets due to a full queue.

–I *interface*
Show information only about this interface; used with an *interval* as described below.

–m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).

–n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.

–p *protocol*
: Show statistics about *protocol*, which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the file */etc/protocols*. A null response typically means that there are no interesting numbers to report. The program will complain if *protocol* is unknown or if there is no statistics routine for it.

–s
: Show per-protocol statistics.

–r
: Show the routing tables. When –s is also present, show routing statistics instead.

–M
: Show the kernel multicast routing tables. When –s is also present, show multicast routing statistics instead.

–f *address_family*
: Limit statistics or address control block reports to those of the specified *address family*. The following address families are recognized: *inet*, for **AF_INET**, and *unix*, for **AF_UNIX**. (*ns*, for AF_NS is not currently supported.)

–u
: A synonym for –f unix.

The arguments, *system* and *core* allow substitutes for the defaults "/unix" and "/dev/kmem".

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the –n option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet "dot format," refer to *inet*(3N). Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), whether the route is to a gateway ("G") or a host ("H"), whether the route was created dynamically by a redirect ("D"), and whether the route has been modified by a redirect ("M"). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the –I option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

hosts(4), networks(4), protocols(4), services(4)

BUGS

The notion of errors is ill-defined.

NAME

newform – change the format of a text file

SYNOPSIS

newform [–**s**] [–**i**tabspec] [–**o**tabspec] [–**b**n] [–**e**n] [–**p**n] [–**a**n] [–**f**] [–**c**char] [–**l**n] [files]

DESCRIPTION

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for –s, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "–e15 –l60" will yield results different from "–l60 –e15". Options are applied to all *files* on the command line.

–**s** Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform –s –i –l –a –e file-name

–**i***tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be ––, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to –8. A *tabspec* of –0 expects no tabs; if any are found, they are treated as –1.

–o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **–i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **–8**. A *tabspec* of **–0** means that no spaces will be converted to tabs on output.

–b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **–l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **–b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:
newform –l1 –b7 file-name

–e*n* Same as **–b***n* except that characters are truncated from the end of the line.

–p*n* Prefix *n* characters (see **–c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

–a*n* Same as **–p***n* except characters are appended to the end of a line.

–f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **–o** option. If no **–o** option is specified, the line which is printed will contain the default specification of **–8**.

–c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

–l*n* Set the effective line length to *n* characters. If *n* is not entered, **–l** defaults to 72. The default line length without the **–l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **–i** to expand tabs to spaces).

The **–l1** must be used to set the effective line length shorter than any existing line in the file so that the **–b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not –s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 – normal execution
1 – for any error

SEE ALSO

csplit(1), tabs(1).
fspec(4) in the *Programmer's Reference Manual.*

BUGS

newform normally only keeps track of physical characters; however, for the **–i** and **–o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **–i—** or **–o—**).

If the **–f** option is used, and the last **–o** option specified was **–o—**, and was preceded by either a **–o—** or a **–i—**, the tab specification format line will be incorrect.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than $ (default) and has not exported PS1. After an invocation of *newgrp* , successful or not, their PS1 will now be set to the default prompt string $. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is not demanded if any of the following are true:

the caller is superuser,

the caller's group id (from /etc/passwd) matches the group's id, or

the caller is a member of the group (i.e. listed in the fourth field of the group entry in /etc/group).

If none of the above conditions are met and the group has a password, the user is asked for it. If the group does not have a password, the requested group-change is denied.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), multgrps(1), sh(1) in the *User's Reference Manual*.
group(4), passwd(4), environ(5) in the *Programmer's Reference Manual*.

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news – print news items

SYNOPSIS

news [**–a**] [**–n**] [**–s**] [items]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

–a option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

–n option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

–s option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5) in the *Programmer's Reference Manual.*

NAME

newshost – NeWS network security control.

SYNOPSIS

newshost add [*hosts*]
  or **newshost remove** [*hosts*]
  or **newshost show**

DESCRIPTION

Newshost is a shell command that manipulates the registry of hosts that are allowed to connect to the NeWS server. The identity of the NeWS server whose registry will be manipulated is determined by the NEWSSERVER environment variable. The variable **/NetSecurityWanted** (in the NeWS **systemdict** may be set to false to disable the security mechanism.

newshost add	adds the named hosts to the registry,
newshost remove	removes the named hosts from the registery,
newshost show	prints out a list of the hosts in the registry.

SEE ALSO

4Sight User's Guide, Section 2, "Programming in NeWS."

NAME

nice – run a command at low priority

SYNOPSIS

nice [−increment] command [arguments]

DESCRIPTION

nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **—10**.

SEE ALSO

nohup(1).
nice(2) in the *Programmer's Reference Manual.*

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl – line numbering filter

SYNOPSIS

nl [–**h**type] [–**b**type] [–**f**type] [–**v**start#] [–**i**incr] [–**p**] [–**l**num] [–**s**sep] [–**w**width] [–**n**format] [–**d**delim] file

DESCRIPTION

nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

–**b***type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

–**h***type* Same as –**b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

a	number all lines
t	number lines with printable text only
n	no line numbering
p*string*	number only lines that contain the regular expression specified in *string*.

Default *type* for logical page body is **t** (text lines numbered).

–f*type*	Same as –b*type* except for footer. Default for logical page footer is **n** (no lines numbered).
–v*start#*	*Start#* is the initial value used to number logical page lines. Default is **1**.
–i*incr*	*Incr* is the increment value used to number logical page lines. Default is **1**.
–p	Do not restart numbering at logical page delimiters.
–l*num*	*Num* is the number of blank lines to be considered as one. For example, **–l2** results in only the second adjacent blank being numbered (if the appropriate **–ha**, **–ba**, and/or **–fa** option is set). Default is **1**.
–s*sep*	*Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
–w*width*	*Width* is the number of characters to be used for the line number. Default *width* is **6**.
–n*format*	*Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).
–d*xx*	The delimiter characters specifying the start of a logical page section may be changed from the default characters (\\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the –d and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl –v10 –i10 –d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nm – name list dump of MIPS object files

SYNOPSIS

nm [**–abdefghnopruvxABTV**] [files]

DESCRIPTION

The *nm* command prints formatted listings of the symbol and external sections of the symbol tables for each *file* specified. A *file* can be an object or an archive. If you do not specify a file, *nm* assumes *a.out*.

The *nm* command supports the options listed below. **NOTE:** some options have different meanings when used with the –**A** or –**B** options.

–A	Use AT&T System V format output. This is the default.
–B	Use Berkeley (4.3BSD) format output.
–a	Print debugging information. When used with **–B**, *nm* uses **–A** format output but with **–B** default ordering and value field radix.
–b	Print the value field in octal.
–d	Print the value field in decimal. This is the default value field radix for **–A**.
–e	Print externals and statics only.
–f	Produce full output. *nm* still accepts this old option, but ignores it.
–g	When used with **–B** print only globally-visible names. When used with **–A** this has no effect.
–h	Do not print headers.
–n	When used alone or with **–A**, sort external symbols by name. This is the default sort order for **–B**. When used with **–B**, sort all symbols by value.
–o	When used alone or with **–A**, print the value field in octal. When used with **–B**, prepend the filename to each symbol. This is useful for *grep*ping through *nm* of libraries.
–p	When used with **–B**, print symbols as they are found in the file. This is the default ordering for **–A**.
–r	Reverse the sense of a value or name sort.
–u	Print only undefined symbols.

—v	Sort external symbols by value.
—x	Print value field in hexadecimal. This is the default value field radix for –**B**.
—T	Truncate long names, inserting a '*' as the last printed character.
—V	Print version information on stderr.

The –**A** System V format (default) and the –**B** Berkeley format specified with –a debugging output provide an expanded listing with these fields:

Name	The symbol or external name.
Value	The value field for the symbol or external, usually an address or interesting debugging information.
Class	The symbol type.
Type	The symbol's language declaration.
Size	Unused.
Index	The symbol's index field.
Section	The symbol's storage class.

NOTE: Every effort was made to map the each field's functionality into System V nomenclature.

The –**B** Berkeley format produces the *address* or *value* field followed by a *letter* showing what section the symbol or external is in and the *name* of the symbol or external.

These section letters describe the information that *nm* generates:

N	Nil storage class, compiler internal usage.
T	External text.
t	Local text.
D	External initialized data.
d	Local initialized data.
B	External zeroed data.
b	Local zeroed data.
A	External absolute.
a	Local absolute.
U	External undefined.

G	External small initialized data.
g	Local small initialized data.
S	External small zeroed data.
s	Local small zeroed data.
R	External read only.
r	Local read only.
C	Common.
E	Small common.
V	External small undefined.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out.** If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out.** If standard output is redirected by the user, then *nohup* will redirect standard error to the same destination as standard output.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file.* If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1),
signal(2) in the *Programmer's Reference Manual.*

WARNINGS

In the case of the following command

```
nohup command1; command2
```

nohup applies only to command1. The command

```
nohup (command1; command2)
```

is syntactically incorrect.

NAME

npri – modify the scheduling priority of a process

SYNOPSIS

npri [**–h** ndpri] [**–n** nice] [**–t** slice] [**–p** pid]
npri [**–h** ndpri] [**–n** nice] [**–t** slice] [cmd args ...]

DESCRIPTION

This command allows the super-user to modify the scheduling priority of a process or to create a new process with a specific priority. For a detailed description of how the parameters affect the scheduling of a process, please see the *schedctl*(2) manual page. If *npri* is invoked without reference to a specific process or command to execute, it simply invokes a copy of the user's shell, as specified by the SHELL environment variable, with the scheduling characteristics specified.

OPTIONS

The following options are supported:

–h *ndpri* This option sets the *non-degrading* priority of a process. Processes with a non-degrading priority are not affected by normal UNIX priority aging. Specifying an *ndpri* of 0 will cancel the existing non-degrading priority of a process. The fixed priorities that can be assigned range from 30 to 255. The range 40 to 127 covers normal interactive jobs. Fixed priorities in the range of 30 to 39 give the process better priority than any normal interactive job. Care should be used when assigned fixed priorities in this range, since such a process is not preemptible by normal interactive processes. Fixed priorities in the range 128 to 255 give the process worse priority than any normal interactive job. Such processes will only get scheduled by a cpu when there are no normal processes that are runnable. Refer to *schedctl*(2) for more details.

–n *nice* This option sets the absolute nice value of a process. The nice value affects how normal UNIX priority calculations are made. The allowable range of nice values is 0 to 39. Refer to *nice*(2) for more information.

–t *slice* This option sets the time-slice for the process, in terms of the basic kernel clock frequency (called *ticks*). The 4D Series machines use a clock frequency of 100 hertz, so one *tick* is 10 milliseconds. For example, a slice value of 3 gives a 30 millisecond time slice. The time slice value will be inherited by any new children of the process.

−p *pid* This option names a specific process whose priority is to be altered. This makes it possible to change priorities of a running process.

SEE ALSO

nice(2), schedctl(2).

NAME

nslookup – query name servers interactively

SYNOPSIS

nslookup [*–option ...*] [*host-to-find* | – [*server*]]

DESCRIPTION

Nslookup is a program to query Internet domain name servers. Nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query the name server for information about various hosts and domains or print a list of hosts in the domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

ARGUMENTS

Interactive mode is entered in the following cases:

a) when no arguments are given (the default name server will be used),

b) when the first argument is a hyphen (–) and the second argument is the host name or Internet address of a name server.

Non-interactive mode is used when the name or Internet address of the host to be looked up is given as the first argument. The optional second argument specifies the host name or address of a name server.

The options listed under the ''set'' command below can be specified on the command line if they precede the arguments and are prefixed with a hyphen. For example, to change the default query type to host information, and the default timeout to 10 seconds, type:

```
nslookup –query=hinfo  –timeout=10
```

INTERACTIVE COMMANDS

Commands may be interrupted at any time by typing a control-C. To exit, type a control-D (EOF) or type exit. The command line length must be less than 80 characters. To treat a built-in command as a host name, precede it with an escape character (\). **N.B.** an unrecognized command will be interpreted as a host name.

host [*server*]

Look up information for *host* using the current default server or using *server* if specified. If *host* is an Internet address and the query types is A, the name of the host is returned. If *host* is a name and does not have a trailing period, the default domain name is appended to the name (see **set domain**). To look up a host not in the current domain, append a period to the name.

server *domain*
lserver *domain*

Change the default server to *domain*. **Lserver** uses the initial server to look up information about *domain* while **server** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

root Changes the default server to the server for the root of the domain name space. Currently, the host ns.nic.ddn.mil is used. (This command is a synonym for **lserver ns.nic.ddn.mil.**) The name of the root server can be changed with the **set root** command.

finger [*name*] [> *filename*]
finger [*name*] [>> *filename*]

Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information (see the **set querytype=A** command). *Name* is optional. > and >> can be used to redirect output in the usual manner.

ls [*option*] *domain* [> *filename*]
ls [*option*] *domain* [>> *filename*]

List the information available for *domain*, optionally creating or appending to *filename*. The default output contains host names and their Internet addresses. *Option* can be one of the following:

–t *querytype*
lists all records of the specified type (see *querytype* below).

–a lists aliases of hosts in the domain. synonym for **–t CNAME**.

–d lists all records for the domain. synonym for **–t ANY**.

–h lists CPU and operating system information for the domain. synonym for **–t HINFO**.

–s lists well-known services of hosts in the domain. synonym for **–t WKS**.

When output is directed to a file, hash marks are printed for every 50 records received from the server.

view *filename*

Sorts and lists the output of previous **ls** command(s) with *more*(1).

help

? Prints a brief summary of commands.

exit Exits the program.

set *keyword*[=*value*]

This command is used to change state information that affects the lookups. Valid keywords are:

all Prints the current values of the frequently-used options to set. Information about the current default server and host is also printed.

class=*value*

Change the query class to one of:

IN	the Internet class.
CHAOS	the Chaos class.
HESIOD	the MIT Athena Hesiod class.
ANY	wildcard (any of the above).

The class specifies the protocol group of the information. (Default = IN, abbreviation = cl)

[no]debug

Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer.
(Default = nodebug, abbreviation = [no]deb)

[no]d2 Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.
(Default = nod2)

domain=*name*

Change the default domain name to *name*. The default domain name is appended to all lookup requests if the **defname** option has been set and the request does not end with a period. The domain search list is set to parents of the default domain if it has at least two components in its name. For example, if the default domain is CC.Berkeley.EDU, the search list is CC.Berkeley.EDU and Berkeley.EDU.
(Default = value in hostname, /usr/etc/resolv.conf or LOCALDOMAIN, abbreviation = do)

[no]defname

Append the default domain name to a name that does not have a trailing dot.
(Default = defname, abbreviation = [no]def)

[no]search

If defname is set, append the domain names in the domain search list to a request until an answer is received.
(Default = search, abbreviation = [no]sea)

port=*value*

Change the default TCP/UDP name server port to *value*.
(Default = 53, abbreviation = po)

querytype=*value*
type=*value*

Change the type of information returned from a query to one of:

A	the host's Internet address.
CNAME	the canonical name for an alias.
HINFO	the host CPU and operating system type.
MINFO	the mailbox or mail list information.
MX	the mail exchanger.
NS	name server for the named zone.
PTR	the host name if the query is an Internet address otherwise the pointer to other information.
SOA	the domain's "start-of-authority" information.
TXT	the text information.
UINFO	the user information.
WKS	the supported well-known services.

Other types (ANY, AXFR, MB, MD, MF, NULL) are described in the RFC-1035 document.
(Default = A, abbreviations = q, ty)

[no]recurse

Tell the name server to query other servers if it does not have the information.
(Default = recurse, abbreviation = [no]rec)

retry=*number*
: Set the number of retries to *number*. When a reply to a request is not received within a certain amount of time (changed with set **timeout**), the timeout period is doubled and the request is resent. The retry value controls how many times a request is resent before giving up.
(Default = 4, abbreviation = ret)

root=*host*
: Change the name of the root server to *host*. This affects the root command.
(Default = ns.nic.ddn.mil., abbreviation = ro)

timeout=*number*
: Change the initial timeout interval for waiting for a reply to *number* seconds. Each retry doubles the timeout period.
(Default = 5 seconds, abbreviation = ti)

[no]vc
: Always use a virtual circuit when sending requests to the server.
(Default = novc, abbreviation = [no]v)

DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

Timed out
: The server did not respond to a request after a certain amount of time (changed with set **timeout**=*value*) and a certain number of retries (changed with set **retry**=*value*).

No response
: No name server is running on the server machine.

No information
: Depending on the query type set with the **set querytype** command, no information about the host was available, though the host name is valid.

Non-existent domain
: The host or domain name does not exist.

Connection refused
Network is unreachable
: The connection to the name or finger server could not be made at the current time. This error commonly occurs with finger requests.

Server failure
: The name server found an internal inconsistency in its database and could not return a valid answer.

Refused
: The name server refused to service the request.

Format error
: The name server found that the request packet was not in the proper format. It may indicate an error in *nslookup*.

FILES

/usr/etc/resolv.conf	initial domain name and name server addresses.
/usr/bsd/nslookup.help	summary of commands.

ENVIRONMENT

HOSTALIASES	file containing host aliases.
LOCALDOMAIN	overrides default domain.

SEE ALSO

resolver(3), resolver(4), named(1M),
RFC-1034 ''Domain Names – Concepts and Facilities''
RFC-1035 ''Domain Names – Implementation and Specification''

O – P

NAME

oclock – display time of day

SYNOPSIS

oclock [-option ...]

DESCRIPTION

Clock simply displays the current time on an analog display

OPTIONS

–fg *foreground color*
: choose a different color for the both hands and the jewel of the clock

–bg *background color*
: choose a different color for the background.

–jewel *jewel color*
: choose a different color for the jewel on the clock.

–minute *minute color*
: choose a different color for the minute hand of the clock.

–hour *hour color*
: choose a different color for the hour hand of the clock.

–backing *{ WhenMapped Always NotUseful }*
: selects an appropriate level of backing store.

–geometry *geometry*
: define the initial window geometry; see *X(1).*

–display *display*
: specify the display to use; see *X(1).*

–bd *border color*
: choose a different color for the window border.

–bw *border width*
: choose a different width for the window border. As the Clock widget changes its border around quite a bit, this is most usefully set to zero.

–noshape
: causes the clock to not reshape itself and ancestors to exactly fit the outline of the clock.

SEE ALSO

X(1), X Toolkit documentation

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Keith Packard, MIT X Consortium

NAME

od – octal dump

SYNOPSIS

od [–bcdosx] [file] [[+]offset[**.**][**b**]]

DESCRIPTION

od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **–o** is default. The meanings of the format options are:

–b	Interpret bytes in octal.
–c	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
–d	Interpret words in unsigned decimal.
–o	Interpret words in octal.
–s	Interpret 16-bit words in signed decimal.
–x	Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.

NAME

odump – dump selected parts of an object file

SYNOPSIS

odump [–acfghilorstFPRzL] files

DESCRIPTION

The *odump* command dumps selected parts of each object *file*.

This command works for object files and archives of object files. It accepts one or more of these options:

–a	Dumps the archive header for each member of the specified archive file.
–f	Dumps each file header.
–g	Dumps the global symbols from the symbol table of a MIPS archive.
–o	Dumps each optional header.
–h	Dumps section headers.
–i	Dumps the symbolic information header.
–s	Dumps section contents.
–r	Dumps relocation information.
–l	Dumps line number information.
–t	Dumps symbol table entries.
–z*name*	Dumps line number entries for the specified function *name*.
–c	Dumps the string table.
–L	Interpret and print the contents of the *.lib* sections.
–F	Dumps the file descriptor table.
–P	Dumps the procedure descriptor table.
–R	Dumps the relative file index table.

The *odump* command accepts these modifiers with the options:

–d *number*	Dumps the section number or a range of sections starting at *number* and ending either at the last section number or the *number* you specify with **+d.**
+d *number*	Dumps sections in the range beginning with the first section or beginning with the section you specify with **–d.**

–n *name*	Dumps information only about the specified *name*. This modifier works with **–h**, **–s**, **–r**, **–l**, and **–t**.
–p	Does not print headers
–t *index*	Dumps only the indexed symbol table entry. You can also specify a range of symbol table entries by using the modifier **–t** with the **+t** option.
+t *index*	Dumps the symbol table entries in the specified range. The range begins at the first symbol table entry or at the entry specified by **–t**. The range ends with the specified indexed entry.
–u	Underlines the name of the file for emphasis.
–v	Dumps information symbolically rather than numerically (for example, Static rather than **0x02**). You can use **–v** with all the options except **–s**.
–z *name,number*	Dumps the specified line number entry or a range of line numbers. The range starts at the *number* for the named function.
+z *number*	Dumps line numbers for a specified range. The range starts at either the *name* or *number* specified by **–z** The range ends with the *number* specified by **+z**.

An option and its modifier may be separated by blanks. The name can be separated from the number that modifies **–z** by replacing the comma with a blank.

The *odump* command tries to format information in a helpful way, printing information in character, hexadecimal, octal, or decimal, as appropriate.

SEE ALSO

a.out(4), ar(4).

NAME

olwm – OPEN LOOK window manager

SYNOPSIS

olwm [**–fp**]

DESCRIPTION

olwm is a window manager for the X Window System, Version 11, that implements parts of the OPEN LOOK TM user interface for window management.

olwm can be started in ''focus-follows-mouse'' mode or ''click-to-focus'' mode. In focus-follows-mouse mode, the window in which the pointer lies has the input focus. In click-to-focus mode, you must click the SELECT (usually the left) mouse button in the window to give it the focus. The focus will stay in that window until you click in another window. The default focus mode is click-to-focus. Start **olwm** with the **–f** option to run in focus-follows-mouse mode.

When **olwm** starts, it reparents all children of the root window.

OPTIONS

–f	Use focus-follows-mouse mode. Default mode is click-to-focus.
–p	Do not reparent windows which are currently displayed. Default is to reparent all windows.

TRADEMARKS

OPEN LOOK is a trademark of AT&T.

NAME

osview – monitor operating system activity data

SYNOPSIS

osview [–in] [–nn] [–unamelist] [–s]

DESCRIPTION

Osview monitors various portions of the activity of the operating system and displays them using the full screen capabilities of the current terminal.

A large number of activity counters are monitored, and the display may be dynamically altered to hide or show only those counters in which the user is interested. It is assumed that the *osview* user is somewhat familiar with the internal workings of an AT&T V.3 based kernel.

Osview lays out as much information as possible in the screen area available. Each data item is grouped similarly to the grouping shown by *sar(1)*. A header line gives the group name, and members of the group are indented below along with the one-second average value over the last interval (or total value over the interval; see below). If a graphics subsystem is not present on the machine being monitored, *osview* suppresses all graphics related statistics in the display.

The **-i** parameter sets the delay between screen updates in seconds. By default, a 5 second rate is used. The **-n** parameter is used to override the default number of lines to use, which is usually the entire size of the display area. This can be useful if the display is called up in a long window, to keep the counters grouped together at the top of the window. The -s option informs *osview* to not reduce relevant values to the average over a second. One second averaging allows instant performance estimates, but may show inaccuracies because of the short interval involved. Finally, the **-u** option allows the specification of a different namelist file for those symbols which must be read from the running kernel. By default, the normal namelist file */unix* is used.

In general, those parameters dealing with data throughput rather than events are presented as the number of bytes involved. For instance, memory usage is reported in bytes, as well as buffer cache traffic. Those parameters dealing with events to the system, such as page fault activity, interrupts or system activity are reported as actual counts. This allows an instant estimate of the activity and throughput of the system.

A group can be suppressed along with all its members to allow hidden groups to be brought into view if the screen area is too small. This is done by moving the cursor over the header line of the group to suppress and typing a suppression character. The cursor may be positioned in any of the standard ways; keyboard arrow keys, the h-j-k-l keys, or the backspace-return-tab keys. *Osview* highlights the line the cursor is on unless the cursor is on the top screen line (which is reserved for status information). When positioned over a group name, typing the D character or one of the *delete* keys on the keyboard will suppress the group. The group name will remain, with an asterisk (*) prefix to indicate that the group has been suppressed. The group may be expanded again by positioning the cursor over the group name and typing the I character or one of the *insert* keys on the keyboard. The *home* key moves the cursor to the *osview* status line.

OVERVIEW

The information which *osview* displays and how to interpret it is given below. See the documentation for *sar(1)* or *gr_osview(1)* for additional information.

Load Average

These counters give a Tenex-style load average over the last minute, 5 minutes or 15 minutes.

CPU Usage

These counters display the proportion of the available processor cycles which were used by each of the following activities. If multiple processors are present, than the CPU number will be added to the header line.

user	- user programs
sys	- system on behalf of user
intr	- interrupt handling
gfxc	- graphics context switching
gfxf	- waiting on graphics input FIFO
idle	- doing nothing

Real Memory

Phys	- physical memory size
Locked	- unpageable memory
Sysdata	- used for filesystem meta-data
Free	- not in use
Delwri	- not yet written to disk
Userdata	- in use holding valid user data
Freeswap	- swap space available

Virtual Memory

vfault	- page faults
pfault	- protection faults
demand	- demand zero and demand fill faults

cw	- copy-on write faults
steal	- page steals
onswap	- page found on swap
oncache	- page found in page cache
onfile	- page read from file
freed	- pages freed by paging daemon
unmodswap	- clean swap page, dirty incore page
unmodfile	- clean file page, dirty incore page

TLB Actions

newpid	- new process ID allocated
tfault	- second level TLB misses
rfault	- reference faults (during paging)
flush	- flush of entire TLB
sync	- cross-processor TLB synchronizations

Swap

swapin	- page swapins
swapout	- page swapouts
bswapin	- bytes swapped in
swapout	- bytes swapped out

TCP

sndtotal	- packets sent
rcvtotal	- packets received
sndbyte	- bytes sent
rcvbyte	- bytes received

Net IF

These counters display the activity on a particular network interface. If multiple interfaces are present, than a separate set of counters is displayed for each interface. The interface name is displayed as part of the header.

Ipackets	- packets received
Opackets	- packets transmitted
Ierrors	- packets received in error
Oerrors	- errors transmitting a packet
collisions	- collisions detected

Interrupts

all	- total interrupts handled
vme	- VMEBus interrupts

Graphics

griioctl	- graphics ioctl's
gintr	- graphics interrupts
swapbuf	- swapbuffer completes
switch	- context switches
fifowait	- wait on FIFO
fifonwait	- wait on FIFO, empty on check

System Activity

syscall	- system calls
read	- read system calls
write	- write system calls
fork	- fork system calls
exec	- exec system calls
readch	- characters read via read()
writech	- characters written via write()
iget	- inode searches

Buffer Cache

lread	- amount of logical buffer reads
bread	- amount of physical buffer reads
%rcache	- read hit ratio on buffer cache
lwrite	- amount of logical buffer writes
bwrite	- amount of physical buffer writes
%wcache	- write hit ratio; negative for write-behind

Wait Ratio

%IO	- waiting on IO
%Swap	- waiting on swap IO
%Physio	- waiting on physical IO

Scheduler

runq	- number of processes on run queue

swapq	- number of processes on swap queue
switch	- context switches

AUTHOR

J. M. Barton

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.z with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called *name*.z already exists;
the .z file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.z use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.z (without destroying *name*.z) use the command:

```
pcat name >nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:

> the file name (exclusive of the .z) has more than 12 characters;
> the file cannot be opened;
> the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

> a file with the ''unpacked'' name already exists;
> if the unpacked file cannot be created.

SEE ALSO

cat(1).

NAME

pandora – login on the graphics console

SYNOPSIS

pandora [**–s**] [**–b** *nframes*]

DESCRIPTION

The *pandora* command is executed at the beginning of each session on the graphics console. It replaces the functionality of the command *login*(1). The choice between *login*(1) and *pandora* is made on the basis of the configuration variable *visuallogin*(4). If this configuration variable is set to on then *pandora* is used in place of *login*(1) Invoking *pandora* while the window manager is running has no effect. The user does not normally run *pandora* directly.

Ordinarily, *pandora* displays a window containing two panes. The top pane contains icons representing each of the users who can log onto the system. The bottom pane contains a text field into which the user can type a user name.

To log in, the user must first enter the correct user name. This can be done by typing it at the keyboard, or by clicking with the left mouse button on the icon representing the user. The user must then click with the left mouse on the button labeled **Login.** The user may also press the Enter key. Quickly clicking twice on the user icon (double-clicking) will have the same effect as typing the user name and pressing the Enter key.

If the user has a password, a second text field is displayed. The user must now type the correct password and either click on the **Login** button or press Enter. If the password is correct, the login process begins. Otherwise, *pandora* returns to its initial state.

The button labeled **Help** may also be used to display some explanatory text.

If the configuration variable *noiconlogin*(4) is set to **on**, the upper pane of the window will not be displayed. The user will be required to type both the user name and the password.

The standard user icons may be replaced with color or black & white images. *pandora* will check in the directories **/usr/local/lib/faces** and **/usr/lib/faces** for files with the same name as the corresponding user. The files must be in SGI image library format, and must be 100 by 100 pixels in size or smaller.

If no image exists for a user, a icon representing the user may be displayed. The file /etc/passwd.sgi contains a list of users who will be displayed with a special icon denoting a "built-in" user.

This file consists of a series of lines, one for each user. Each line contains a list of colon-separated fields. The first field on each line is the name of the user. Additional fields may be provided. If the fields "noshow" or "noicon" are present, the icon will not be displayed on the login panel. The user will be required to type the full username in order to log in.

Users whose home directory does not exist will not be displayed as icons.

The login process for the graphics console may be automated by configuring the system to autologin. The autologin process, when enabled, bypasses the *pandora* login when a valid entry is read from /etc/autologin. The entry is then used as the identity for the session. Autologin is disabled by *login* once it has successfully initiated a session on the graphics console. Autologin is enabled during boot time if the /etc/autologin file exists.

To perform a textport login rather than enter the window manager, the user may type the string NOGRAPHICS after the user name.

For more information on how the initial environment is configured, see *login*(1).

OPTIONS

−s Used to select whether or not icons will be displayed.

−b *nframes*
Set the screen blanking time to *nframes* screen refreshes. See *blanktime*(1) for details on screen blanking.

NOTES

Autologin is controlled by the existence of the **/etc/autologin.on** file. The file is normally created at boot time to automate the login process and then removed by *login* or *pandora* to disable the autologin process for succeeding terminal sessions.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/passwd.sgi	local SGI user annotation file
/etc/profile	system profile (for *sh*(1))
.profile	user's login profile (for *sh*(1))
/etc/cshrc	system-wide *csh*(1) commands
.login	user's *csh*(1) commands for login shell
.cshrc	user's *csh*(1) commands for every shell
/etc/autologin	autologin user identity
/etc/autologin.on	enable autologin at boot time

SEE ALSO

blanktime(1), csh(1), login(1), mail(1), newgrp(1), sh(1) in the *IRIX User's Reference Manual*.
chkconfig(1M), su(1M) in the *IRIX System Administrator's Reference Manual*.
passwd(4), profile(4), environ(5), noiconlogin(4), visuallogin(4) in the *IRIX Programmer's Reference Manual*.

NAME

passwd – change login password and password attributes

SYNOPSIS

passwd [*name*]

passwd [**–l** | **–d**] [**–n** *min*] [**–f**] [**–x** *max*] *name*

passwd **–s** [**–a**]

passwd **–s** [*name*]

DESCRIPTION

The *passwd* command changes the password or lists password attributes associated with the user's login *name*. Additionally, super-users may use *passwd* to install or change passwords and attributes associated with any login *name*.

When used to change a password, *passwd* prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has ''aged'' sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming ''aging'' is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

> Each password must have at least six characters. Only the first eight characters are significant.
>
> Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, ''alphabetic'' means upper and lower case letters.
>
> Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.
>
> New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password. (This differs from **passwd -d** because the "password" prompt will still be displayed.)

Any user may use the **-s** option to show password attributes for his or her own login *name*.

The format of the display will be:

name status uid gid directory shell mm/dd/yy *min max*

or, if password aging information is not present,

name status uid gid directory shell

or, if yellow page entry,

name status directory shell

where

name	The login ID of the user.
status	The password status of *name*: "PS" stands for passworded or locked, "LK" stands for locked, and "NP" stands for no password.
uid	Numerical user id
gid	Numerical group id
directory	Initial working directory
shell	program to use as Shell when the user logs in.
mm/dd/yy	The date password was last changed for *name*. (Note that all password aging dates are determined using Greenwich Mean Time and, therefore, may differ by as much as a day in other time zones.)
min	The minimum number of days required between password changes for *name*.
max	The maximum number of days the password is valid for *name*.

Only a super-user can use the following options:

–l	Locks password entry for *name*.
–d	Deletes password for *name*. The login *name* will not be prompted for password.
–n	Set minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*. If *min* is greater than *max*, the user may not change the password. Always use this option with the –**x** option, unless *max* is set to –1 (aging turned off). In that case, *min* need not be set.
–x	Set maximum field for *name*. The *max* field contains the number of days that the password is valid for *name*. The aging for *name* will be turned off immediately if *max* is set to -1. If it is set to 0, then the user is forced to change the password at the next login session and aging is turned off.
–a	Show password attributes for all entries. Use only with –s option; *name* must not be provided.
–f	Force the user to change password at the next login by expiring the password for *name*.

FILES

/etc/passwd, /etc/opasswd, /ctc/.pwd.lock

SEE ALSO

login(1).
crypt(3C), passwd(4) in the *Programmer's Reference Manual.*
passmgmt(1M), id(1M), su(1M) in the *System Administrator's Reference Manual.*

DIAGNOSTICS

The **passwd** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.
6	Invalid argument to option.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste **–d** list file1 file2 ...
paste –s [**–d** list] file1 file2 ...

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

The meanings of the options are:

-d Without this option, the new-line characters of each but the last file (or last line in case of the –s option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **–d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no –s option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *–d* "\\\\").

-s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with –d option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

– May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste –d" " –	list directory in one column
ls | paste – – – –	list directory in four columns
paste –s –d"\t\n" file	combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for –s option, no more than 12 input files may be specified.

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [*–number*] [**–p** *string*] [**–cefns**] [*+linenumber*] [*+/pattern/*] [files...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name – and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

—number
: An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

—p *string*
: Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

—c
: Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

—e
: Causes *pg* *not* to pause at the end of each file.

—f
: Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *–f* option inhibits *pg* from splitting lines.

—n
: Normally, commands must be terminated by a *<newline>* character. This option causes an automatic end of command as soon as a command letter is entered.

–s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+*linenumber*
Start up at *linenumber*.

+/*pattern*/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the –*n* option is specified.

i/*pattern*/
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
i?*pattern*?
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i*w Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a <*newline*>, even if the *–n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or Q Quit *pg*.

!*command*
Command is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the *–n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1).
terminfo(4) in the *Programmer's Reference Manual*.

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

NAME

pixie – add profiling code to a program

SYNOPSIS

pixie in_prog_name [options]

DESCRIPTION

Pixie reads an executable program, partitions it into basic blocks, and writes an equivalent program containing additional code that counts the execution of each basic block. (A basic block is a region of the program that can be entered only at the beginning and exited only at the end). Pixie also generates a file containing the address of each of the basic blocks.

When you run the pixie-generated program, it will (provided it terminates normally or via a call to *exit*(2)) generate a file containing the basic block counts. The name of the file is that of the original program with any leading directory names removed and ".Counts" appended. If the program calls *sproc*(2) then multiple files will be generated, one for each process in the share group. In this case, each ".Counts" file will additionally have the 5 digit process id appended to the name. *Prof*(1) and *pixstats*(1) can analyze these files and produce a listing of profiling data.

–o *out_prog_name*
: Specify a name for the translation. Default is to remove any leading directory names from the in_prog_name and append ".pixie".

–bbaddrs *name*
: Specify a name for the file of basic block addresses. Default is to remove any leading directory names from the in_prog_name and append ".Addrs".

–[no]quiet
: [Permits] or suppresses messages summarizing the binary-to-binary translation process. Default: **–noquiet.**

–[no]dwops
: Controls translation of double-word load/store instructions so that binaries using these instructions can be run on old processors. Default: **–nodwops** (translate).

–[no]textdata
: Controls whether pixie puts the original text into the translated output. This is required to correctly translate programs with data in the text section (e.g. f77 format statements in some compiler releases). Default: **–textdata** (include original text).

–[no]idtrace
: [Disable] or enable tracing of instruction and data memory references. **–idtrace** is equivalent to using both **–itrace** and **–dtrace** together. Default: **–noidtrace**

–[no]itrace
: [Disable] or enable tracing of instruction memory references. Default: **–noitrace**

–[no]dtrace
: [Disable] or enable tracing of data memory references. For the moment, **–dtrace** requires **–itrace**. Default: **–nodtrace**

–[no]oldtrace
: [Disable] or enable the old memory reference trace format. Default: **–oldtrace.**

–idtrace_sample *number*
: Record only 1 out of every *number* memory reference chunks. (This feature not yet implemented.)

–idtrace_file *number*
: Specify a UNIX file descriptor number for the trace output file. Default: 19.

SEE ALSO

prof(1), pixstats(1)

NOTES

When *pixie* reports the size of the old and new code, the numbers are accurate. However, the new code size does not include read-only data (including a copy of the original text) put into the *pixie*d text section. Therefore *size*(1) on the *pixie* output reports a much larger text size.

BUGS

The handler function address to the signal system calls is not translated, so programs that receive signals will not work pixified.

Pixified code is substantially larger than the original code. Conditional branches that used to fit in the 16-bit branch displacement field may no longer fit, generating a pixie error.

pixie does not support shared libraries and will exit with an error message if it is invoked on a program loaded with one.

prof(1) cannot handle stripped programs. *pixie* will give a message warning the user of this if it is invoked on a stripped executable.

The −**itrace** and −**dtrace** options are not supported for programs that use *sproc*(2).

NAME

pixstats — analyze program execution

SYNOPSIS

pixstats program [options]

DESCRIPTION

Pixstats analyzes a program's execution characteristics. To use *pixstats*, first use *pixie*(1) to translate and instrument the executable object module for the program. Next, execute the translation on an appropriate input. This produces a *.Counts* file. Finally, use *pixstats* to generate a detailed report on opcode frequencies, interlocks, a mini-profile, and more.

–cycle *ns*	Assume a *ns* cycle time when converting cycle counts to seconds.
–r2010	Use r2010 floating point chip operation times and overlap rules. This is the default.
–r2360	Use r2360 floating point board operation times and overlap rules.
–disassemble	Disassemble and show the analyzed object code.

SEE ALSO

pixie(1), prof(1)

BUGS

Pixstats models execution assuming a perfect memory system. Cache misses etc. will increase execution above the *pixstats* predictions.

NAME

plaid – paint some plaid-like patterns in an X window

SYNOPSIS

plaid [-option ...]

OPTIONS

–b enable backing store for the window

–fg *color*
This option specifies the color to use for the foreground of the window. The default is "white."

–bg *color*
This option specifies the color to use for the background of the window. The default is "black."

–bd *color*
This option specifies the color to use for the border of the window. The default is "white."

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–geometry *geometry*
define the initial window geometry; see *X(1)*.

–display *display*
specify the display to use; see *X(1)*.

DESCRIPTION

Plaid displays a continually changing plaid-like pattern in a window.

SEE ALSO

X(1)

BUGS

There are no known bugs. There are lots of lacking features.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

NAME

pmake, smake – create programs in parallel

SYNOPSIS

pmake [**–d** *what*] [**–f** *makefile*] [**–h**] [**–i**] [**–k**] [**–l**] [**–n**] [**–p** *#*] [**–q**] [**–r**] [–s] [**–t**] [**–v**] [**–B**] [–C] [**–D** *variable*] [**–I** *directory*] [**–J** *#*] [–M] [–P] [–V] [–W] [*VAR1=value1*] [*VAR2=value2*...] [*targ1*] [*targ2* ...]

ARGUMENTS

–d	*what*	Specify what modules should print debugging information. *what* is a string of letters from the following set: a, c, d, j, m, p, r, s, t, v.
–f	*makefile*	Specify a different makefile to read than the standard ''Makefile'' or ''makefile''. If *makefile* is "–", standard input is read.
–h		Prints out help information and default values.
–i		''Ignore errors'' — ignore non-zero exit statuses of commands.
–k		''Keep-going'' — if an error is encountered, keep working on those parts of the input graph that are not affected by the error.
–l		Create a lock file (called ''LOCK.make'') to prevent other people from executing *PMake* in the same directory. Useful when simultaneous makes in the same place can be disastrous for the final product (too many cooks and all that). Note that this locking will not prevent *you* from invoking *PMake* twice in the same place — if you own the lock file, *PMake* will warn you about it but continue to execute.
–n		''No execute'' — do not execute commands. Just print the ones that would be executed.
–p	#	Tell *PMake* if and when to print the input graph. The number is the bitwise OR of the numbers 1 and 2. 1 means print the graph before making anything and 2 means print the graph after making everything. 3 means do both.
–q		''Query'' — do not execute any commands. Just exit 0 if the given target(s) is (are) up to date and exit non-zero otherwise.

−r		"Remove built-in rules" — do not parse the built-in rules given in the system makefile.
−s		"Silence" — do not echo commands as they are executed.
−t		"Touch targets" — rather than executing the commands to create a target, just change its modification time so it appears up-to-date. This is dangerous.
−v		"System V" — invokes compatibility functions suitable for acting like the System V (IRIX) version of *make*(1). This implies **−B**, and **−V**. Automatically set when *PMake* is invoked as *smake*.
−B		"Backwards-compatible" — performs as much like *make*(1) as possible (including executing a single shell per command and expanding variables as *make* did) while still performing in parallel.
−C		"Non-compatible" — turns off all compatibility specified up to the point at which −C is encountered.
−D	*variable*	Defines the given variable to be **1** in the global context.
−I	*directory*	Specify another directory in which to look for #include'd makefiles. This flag may be repeated as many times as necessary.
−J	#	Specify the maximum number of jobs to run at once.
−M		Be as much like *make*(1) as possible. No parallel execution. Old-style variable expansion. One shell per command, etc.
−P		"Don't use Pipes" — see the section on **OUTPUT**.
−V		"Do old-style variable expansion" — expands an unknown variable to the empty string.
−W		Don't print warning messages.
VAR=value		Set the value of the variable **VAR** to the given value. This supersedes any value assigned to the variable in the makefile. See **VARIABLES**.

smake is equivalent to **pmake** −v.

The flags −x, −X and −L are recognized but ignored by this implementation of *Pmake*.

DESCRIPTION

PMake is a program designed to make the maintenance of other programs much easier. Its input is a ''makefile'' that specifies which files depend on which other files and what to do about files that are ''out-of-date.'' PMake's most important feature is its ability to run several different jobs at once, making the creation of systems considerably faster. It also has a great deal more functionality than *make*(1).

PMake is almost fully-compatible with *make*(1). The main differences are:

1) The *PMake* variable substitution, as described below, is very different and will cause problems unless the makefile is converted or the −V flag is given or when *PMake* is invoked as *smake*.

2) Because *PMake* creates targets in parallel, certain sequences which depend on the sources of a target being created sequentially will fail. For example:

   ```
   prod : $(PROGRAM) clean
   ```

   This is liable to cause some of the object files to be removed after having been created during the current invocation (or, at the very least, the object files will not be removed when the program has been made), leading to errors in the final linking stage. This problem cannot even be avoided by limiting the maximum concurrency to one, since the traversal of the dependency graph is done in a breadth-first, rather than a depth-first way. For *make*(1) behavior, rewrite the makefile, or give *PMake* the −M flag.

3) *PMake* forks only one shell to execute the commands to re-create a target. This means that changes of directory, environment, etc., remain in effect throughout the creation process. It also allows for a more natural entry of shell loop constructs without the need for backslashes and semi-colons required by the one-shell-per-command paradigm used by *make*(1). It is possible to have *PMake* execute each command in a single shell by giving it the −B flag.

4) *PMake* strips the leading directory from the files in a target's local variables, unlike *make*(1). For example, *PMake* looks in the current directory for file.c when making gen/file.o:

   ```
   default:  gen/file.o
   .c.o:
             cc −c  −o  $@  $<
   ```

   To have *PMake* look in the gen directory for file.c, add a ''.PATH: gen'' target to the makefile. The .PATH target, which is described below, is ignored by *make*(1).

5) *PMake* can interpret a comment line that begins with ``# if'' as a conditional statement. Duplicate the comment character before the line to avoid a warning message.

6) *PMake* doesn't have *make*(1)'s tilde rules for SCCS files.

MAKEFILES

If you don't specify a makefile to read, *PMake* looks for **Makefile** in the current directory. If that file does not exist, it looks for **makefile.** (This search order is reversed for *smake* or when using the **–M** flag.)

There are four basic types of lines in a makefile:

1) File dependency specifications

2) Creation commands

3) Variable assignments

4) Comments, include statements and conditional directives

Any line may be continued over multiple lines by ending it with a backslash. The backslash, following newline and any initial white-space on the following line are compressed into a single space.

DEPENDENCY LINES

On a dependency line, there are targets, sources and an operator. The targets ``depend'' on the sources and are usually created from them. Any number of targets and sources may be specified on a dependency line. All the targets in the line are made to depend on all the sources. If you run out of room, use a backslash at the end of the line to continue onto the next one.

Any file may be a target and any file may be a source, but the relationship between them is determined by the ``operator'' that separates them. Three operators are defined:

:	A target on the line is considered ``out-of-date'' if any of its sources has been modified more recently than the target. Sources for a target accumulate over lines when this operator is used.
!	Targets will always be re-created, but this will not happen until all of its sources have been examined and re-created, if necessary. Sources accumulate over lines as for the colon.
::	Much like the colon, but acts like the ! operator if no sources are specified. In addition sources do not accumulate over lines. Rather, the commands associated with the line (see below) are executed only if the target is out-of-date with respect to the sources on that line only. In

addition, the target will not be removed if *PMake* is interrupted, unlike for the other two operators.

For example:

```
a :  a.o b.o c.o
b !  d.o e.o
c :: f.o
        command1
a :  g.o
b !  h.o
c ::
        command2
```

specifies that a depends on a.o, b.o, c.o and g.o and will be remade only if out-of-date with respect to these four files. b depends on d.o, e.o and h.o and will always be remade, but only after these three files have been remade. c will be remade with command1 if it is out-of-date with respect to f.o, as for the colon operator, while command2 will always be executed.

Targets and sources may also contain standard shell wildcard characters (?, *, [and {}), but the ?, *, [and] characters may only be used in the final component of the target or source. If a target or source contains only curly braces and no other wildcard characters, it need not describe an existing file. Otherwise, only existing files will be used. For example, the pattern

```
{a,b,c}.o
```

will expand to

```
a.o b.o c.o
```

regardless of whether these three files exist, while

```
[abc].o
```

will only expand to this if all three files exist. The resulting expansion is in directory order, not alphabetically sorted as in the shell.

COMMANDS

Associated with each target is a series of shell commands, collectively called a script. The creation script for a target should immediately follow the dependency line for that target. Each of the commands in this script *must* be preceded by a tab character.

While any given target may appear on more than one dependency line, only one of these dependency lines may be followed by a creation script, unless the "::" operator is used.

One helpful feature of *PMake* is the ability to delay execution of a target's commands until everything else has been done. To do this, make one of the commands for the target be just ''...'' (an ellipsis) on a line by itself. The ellipsis itself won't be executed, of course, but any commands in the target's script that follow the ellipsis will be saved until *PMake* is done processing everything it needs to process. If you were to say,

```
a.o : a.c
        cc -c a.c
        ...
        @echo "All done"
```

Then the command ''echo "All done"'' would execute once everything else had finished. Note that this will only happen if ''a.o'' is found to be out-of-date.

There is another way in which makefile shell commands differ from regular shell commands, as illustrated in the previous example. The first two characters after the initial tab (and any other white-space) are treated specially. If they are any combination of '@' and '–', (''@'', ''@–'', ''–@'' or ''–''), they cause *PMake* to do different things.

In most cases, shell commands are printed to the screen before they're actually executed. This is to keep you informed of what's going on. If an '@' appears, however, this echoing is suppressed. In the case of the echo command, above, this makes sense. It would look silly to see

```
echo "All done"
All done
```

so *PMake* allows you to avoid that (this sort of echo control is only available if you use the Bourne or C shells to execute your commands, since the commands are echoed by the shell, not by *PMake*).

The other special character is the '–'. Shell commands exit with a certain ''exit status.'' Normally this status will be 0 if everything went ok and non-zero if something went wrong. For this reason, *PMake* will consider an error to have occurred if one of the commands it invokes returns a non-zero status. When it detects an error, its usual action is to stop working, wait for everything in process to finish, and exit with a non-zero status itself. This behavior can be altered, however, by means of **–i** or **–k** arguments, or by placing a '–' at the front of the command. (Another quick note: the decision of whether to abort a target when one of its shell commands returns non-zero is left to the shell that is executing the commands. Some shells allow this ''error-checking'' to be switched on and off at will while others do not.)

VARIABLES

PMake has the ability to save text in variables to be recalled later at your convenience. Variables in *PMake* are used much like variables in *sh*(1) and, by tradition, consist of all upper-case letters. They are assigned- and appended-to using lines of the form

VARIABLE = value
VARIABLE += value

respectively, while being conditionally assigned-to (if not already defined) and assigned-to with expansion by lines of the form

VARIABLE ?= value
VARIABLE := value

Finally,

VARIABLE != command

will execute *command* using the Bourne shell and place the result in the given variable. Newlines are converted to spaces before the assignment is made. This is not intended to be used with commands that produce a large amount of output. If you use it this way, *PMake* will probably deadlock.

Variables are expanded by enclosing the variable name in either parentheses or curly braces and preceding the whole thing with a dollar sign. For example, to set the variable **CFLAGS** to the string "–I/usr/include/bsd –O" place the line

CFLAGS = –I/usr/include/bsd –O

in the makefile and use the word $(CFLAGS) wherever you would like the string "–I/usr/include/bsd –O" to appear. To pass a string of the form "$(*name*)" or "${*name*}" through to the shell (e.g., to tell it to substitute one of its variables), you can use "$$(*name*)" and "$${*name*}", respectively, or, as long as the *name* is not a *PMake* variable, you can just place the string in directly, as *PMake* will not expand a variable it doesn't know, unless it is given one of the three compatibility flags –V, –B, or –M, or invoked as *smake*.

There are two distinct times at which variable substitution occurs: When parsing a dependency line, such substitution occurs immediately upon reading the line. Thus all variables used in dependency lines must be defined before they appear on any dependency line. For variables that appear in shell commands, variable substitution occurs when the command is processed, that is, when it is prepared to be passed to the shell or before being saved for later execution (see **COMMANDS** above).

There are four different types of variables at which *PMake* will look when trying to expand any given variable. They are (in order of decreasing precedence): (1) variables that are defined specific to a certain target. These are the so-called ''local'' variables and will only be used when performing variable substitution on the target's shell script and in dynamic sources (see below for more details), (2) variables that were defined on the command line, (3) variables defined in the makefile and (4) those defined in *PMake*'s environment, as passed by your login shell. An important side effect of this searching order is that once you define a variable on the command line, *nothing* in the makefile can change it.

As mentioned above, each target has associated with it as many as seven ''local'' variables. Four of these variables are always set for every target that must be re-created. Each local variable has a long, meaningful name and a short, one-character name that exists for backwards-compatibility. They are:

.TARGET (@)	The name of the target.
.OODATE (?)	The list of sources for this target that were deemed out-of-date.
.ALLSRC (>)	The list of all sources for this target.
.PREFIX (*)	The file prefix of the file. This contains only the file portion – no suffix or leading directory components.

Three other ''local'' variables are set only for certain targets under special circumstances. These are the ''.IMPSRC'', ''.ARCHIVE'' and ''.MEMBER'' variables. When they are set, how they are used, and what their short forms are detailed in later sections.

In addition, for System V Make compatibility, the variables ''@F'', ''<F'', and ''*F'' are defined as the file parts of the ''@'', ''>'' and ''*'' variables. Likewise, ''@D'', ''<D'', and ''*D'' are directory parts.

Four of these local variables may be used in sources on dependency lines. The variables expand to the proper value for each target on the line. The variables are ''.TARGET'', ''.PREFIX'', ''.ARCHIVE'', and ''.MEMBER''.

In addition, certain variables are set by or have special meaning to *PMake*. The **.PMAKE** (and **MAKE**) variable is set to the name by which *PMake* was invoked, to allow recursive makes to use the same version, whatever it may be. All command-line flags given to *PMake* are stored in the **.MAKEFLAGS** (and **MFLAGS**) variable just as they were given. This variable is also exported to subshells as the **PMAKE** environment variable.

Variable expansion may be modified as for the C shell. A general expansion specification looks like:

> $(*variable*[:*modifier*[:...]])

Each modifier begins with a single character, thus:

M*pattern*
: This is used to select only those words (a word is a series of characters that are neither spaces nor tabs) that match the given *pattern* . The pattern is a wildcard pattern like that used by the shell, where "*" means 0 or more characters of any sort; "?" is any single character; "[abcd]" matches any single character that is either 'a', 'b', 'c' or 'd' (there may be any number of characters between the brackets); **[0-9]** matches any single character that is between '0' and '9' (i.e., any digit. This form may be freely mixed with the other bracket form), and \ is used to escape any of the characters "*", "?", "[" or ":", leaving them as regular characters to match themselves in a word.

N*pattern*
: This is identical to ":M" except it substitutes all words that don't match the given pattern.

S/*search-string*/*replacement-string*/[g]
: Causes the first occurrence of *search-string* in the variable to be replaced by *replacement-string*, unless the "g" flag is given at the end, in which case all occurrences of the string are replaced. The substitution is performed on each word in the variable in turn. If *search-string* begins with a "^", the string must match starting at the beginning of the word. If *search-string* ends with a "$", the string must match to the end of the word (these two may be combined to force an exact match). If a backslash precedes these two characters, however, they lose their special meaning. Variable expansion also occurs in the normal fashion inside both the *search-string* and the *replacement-string* , except that a backslash is used to prevent the expansion of a "$", not another dollar sign, as is usual. Note that *search-string* is just a string, not a pattern, so none of the usual regular-expression/wildcard characters has any special meaning save "^" and "$". In the replacement string, the "&" character is replaced by the *search-string* unless it is preceded by a backslash. You are allowed to use any character except colon or

exclamation point to separate the two strings. This so-called delimiter character may be placed in either string by preceding it with a backslash.

T Replaces each word in the variable expansion by its last component (its ``tail'').

H This is similar to ":T", except that every word is replaced by everything but the tail (the ``head'').

E ":E" replaces each word by its suffix (``extension'').

R This replaces each word by everything but the suffix (the ``root'' of the word).

In addition, *PMake* supports the System V form of substitution

$(*variable:string1=string2*)

where all occurrences of *string1* at the end of each word in the variable expansion are replaced by *string2.*

COMMENTS, INCLUSION AND CONDITIONALS

Makefile inclusion and conditional structures reminiscent of the C compiler have also been included in *PMake*.

Comments begin with a '#' anywhere but in a shell command and continue to the end of the line. The comment character can included in macros if preceded with a backslash (\). If the '#' comes at the beginning of the line, however, the following keywords are recognized and acted on:

#include "*makefile*"
#include <*system makefile*>

This is very similar to the C compiler's file-inclusion facility, right down to the syntax. What follows the **#include** must be a filename enclosed either in double-quotes or angle brackets. Variables will be expanded between the double-quotes or angle-brackets. If angle-brackets are used, the system makefile directory is searched. If the name is enclosed in double-quotes, the including makefile's directory, followed by all directories given via –I arguments, followed by the system directory, is searched for a file of the given name.

If the file is found, *PMake* starts taking input from that file as if it were part of the original makefile.

When the end of the file is reached, *PMake* goes back to the previous file and continues from where it left off. This facility is recursive up to a depth limited only by the number of open files allowed to any process at one time.

include *makefile*

This (non-standard) include syntax is recognized for compatibility with the IRIX *make*(1) command. No search paths are used. The file name, which may contain variables, must be an absolute path of a makefile.

#if [!] *expr* [*op expr* ...]
#ifdef [!] *variable* [*op variable*...]
#ifndef [!] *variable* [*op variable*...]
#ifmake [!] *target* [*op target*...]
#ifnmake [!] *target* [*op target*...]

These are all the beginnings of conditional constructs in the spirit of the C compiler. Conditionals may be nested to a depth of thirty.

In the expressions given above, *op* may be either || (logical OR) or && (logical AND). && has a higher precedence than ||. As in C, *PMake* will evaluate an expression only as far as necessary to determine its value. If the left side of an && is false, the expression is false and vice versa for ||. Parentheses may be used as usual to change the order of evaluation.

One other boolean operator is provided: ! (logical negation). It is of a higher precedence than either the AND or OR operators, and may be applied in any of the "if" constructs, negating the given function for "#if" or the implicit function for the other four.

Expr can be one of several things. Four functions are provided, each of which takes a different sort of argument.

The function **defined** is used to test for the existence of a variable. Its argument is, therefore, a variable name. Certain variable names (e.g., "IRIX", "SYSV", and "unix") are defined in the system makefile (see **FILES**) to specify the sort of system on which *PMake* is being run. These are intended to make makefiles more portable. Any variable may be used as the argument of the **defined** function.

The **make** function is given the name of a target in the makefile and evaluates to true if the target was given on *PMake*'s command-line or as a source for the **.MAIN** target before the line containing the conditional.

The **exists** function takes a file name, which file is searched for on the system search path (as defined by **.PATH** targets (see below)). It evaluates true if the file is found.

The function **empty** takes a variable expansion specification (minus the dollar sign) as its argument. If the resulting expansion is empty, this evaluates true.

Expr can also be an arithmetic or string comparison, with the left-hand side being a variable expansion. The standard C relational operators are allowed, and the usual number/base conversion is performed, with the exception that octal numbers are not supported. If the right-hand side of a "==" or "!=" operator begins with a quotation mark, a string comparison is done between the expanded variable and the text between the quotation marks. If no relational operator is given, it is assumed that the expanded variable is to be compared against 0, i.e., it is interpreted as a boolean, with a 0 value being false and a non-zero value being true.

When, in the course of evaluating one of these conditional expressions, *PMake* encounters some word it does not recognize, it applies one of either *make* or *defined* to it, depending on the form of ''if'' used. For example, ''#ifdef'' will apply the *defined* function, while ''#ifnmake'' will apply the negation of the *make* function.

If the expression following one of these forms evaluates true, the reading of the makefile continues as before. If it evaluates false, the following lines are skipped. In both cases, this continues until either an **#else** or an **#endif** line is encountered.

#else
The #else, as in the C compiler, causes the sense of the last conditional to be inverted and the reading of the makefile to be based on this new value, i.e., if the previous expression evaluated true, the parsing of the makefile is suspended until an #endif line is read. If the previous expression evaluated false, the parsing of the makefile is resumed.

#elif [!] *expr* [*op expr* ...]
#elifdef [!] *variable* [*op variable*...]
#elifndef [!] *variable* [*op variable*...]
#elifmake [!] *target* [*op target*...]
#elifnmake [!] *target* [*op target*...]
The ''elif'' constructs are a combination of ''else'' and ''if,'' as the name implies. If the preceding ''if'' evaluated false, the expression following the ''elif'' is evaluated and the lines following it are read or ignored the same as for a regular ''if.'' If the preceding ''if'' evaluated true, however, the ''elif'' is ignored and all following lines until the ''endif'' (see below) are ignored.

#endif
#endif is used to end a conditional section. If lines were being skipped, the reading of the makefile resumes. Otherwise, it has no effect (the makefile continues to be parsed as it was just before the #endif was encountered).

#undef

Takes the next word on the line as a global variable to be undefined (only undefines global variables, not command-line variables). If the variable is already undefined, no message is generated.

TARGET ATTRIBUTES

In *PMake*, files can have certain "attributes." These attributes cause *PMake* to treat the targets in special ways. An attribute is a special word given as a source to a target on a dependency line. The words and their functions are given below:

.DONTCARE	If a target is marked with this attribute and *PMake* can't figure out how to create it, it will ignore this fact and assume the file isn't really needed or actually exists and *PMake* just can't find it.
.EXEC	This causes the marked target's shell script to always be executed (unless the **−n** or **−t** flag is given), but appear invisible to any targets that depend on it.
.IGNORE	Giving a target the **.IGNORE** attribute causes *PMake* to ignore errors from any of the target's commands, as if they all had '−' before them.
.INVISIBLE	This allows you to specify one target as a source for another without the one affecting the other's local variables.
.JOIN	This forces the target's shell script to be executed only if one or more of the sources was out-of-date. In addition, the target's name, in both its **.TARGET** variable and all the local variables of any target that depends on it, is replaced by the value of its **.ALLSRC** variable. Another aspect of the .JOIN attribute is it keeps the target from being created if the **−t** flag was given.
.MAKE	The **.MAKE** attribute marks its target as being a recursive invocation of *PMake* . This forces *PMake* to execute the script associated with the target (if it is out-of-date) even if you gave the −n or −**t** flag.
.NOTMAIN	Normally, if you do not specify a target to make in any other way, *PMake* will take the first target on the first dependency line of a makefile as the target to create. Giving a target this attribute keeps it from this fate.

.PRECIOUS	When *PMake* is interrupted, it will attempt to clean up after itself by removing any half-made targets. If a target has this attribute, however, *PMake* will leave it alone
.SILENT	Marking a target with this attribute keeps its commands from being printed when they're executed.
.USE	By giving a target this attribute, you turn the target into *PMake*'s equivalent of a macro. When the target is used as a source for another target, the other target acquires the commands, sources and attributes (except **.USE**) of the source. If the target already has commands, the **.USE** target's commands are added to the end. If more than one .USE-marked source is given to a target, the rules are applied sequentially.

The following target attributes are recognized but not implemented on IRIX: .EXPORT, .EXPORTSAME, and .NOEXPORT.

SPECIAL TARGETS

As there were in *make*(1), so there are certain targets that have special meaning to *PMake* . When you use one on a dependency line, it is the only target that may appear on the left-hand-side of the operator. The targets are as follows:

.BEGIN	Any commands attached to this target are executed before anything else is done. You can use it for any initialization that needs doing.
.DEFAULT	This is sort of a .USE rule for any target (that was used only as a source) that *PMake* can't figure out any other way to create. Only the shell script is used. The **.IMPSRC** variable of a target that inherits **.DEFAULT**'s commands is set to the target's own name.
.END	This serves a function similar to **.BEGIN**: commands attached to it are executed once everything has been re-created (so long as no errors occurred). It also serves the extra function of being a place on which *PMake* can hang commands you put off to the end. Thus the script for this target will be executed before any of the commands you save with the "...".
.IGNORE	This target marks each of its sources with the **.IGNORE** attribute. If you don't give it any sources, then it is like giving the **–i** flag.

.INCLUDES The sources for this target are taken to be suffixes that indicate a file that can be included in a program source file. The suffix must have already been declared with **.SUFFIXES** (see below). Any suffix so marked will have the directories on its search path (see **.PATH**, below) placed in the **.INCLUDES** variable, each preceded by a **–I** flag. The **.h** suffix is already marked in this way in the system makefile.

.INTERRUPT When *PMake* is interrupted, it will execute the commands in the script for this target, if it exists.

.LIBS This does for libraries what **.INCLUDES** does for include files, except the flag used is **–L**, as required by those linkers that allow you to tell them where to find libraries. The variable used is **.LIBS**.

.MAIN If you didn't give a target (or targets) to create when you invoked *PMake* , it will take the sources of this target as the targets to create.

.MAKEFLAGS This target provides a way for you to always specify flags for *PMake* when the makefile is used. The flags are just as they would be typed to the shell, though the **–f** and **–r** flags have no effect.

.NULL This allows you to specify what suffix *PMake* should pretend a file has if, in fact, it has no known suffix. Only one suffix may be so designated. The last source on the dependency line is the suffix that is used (you should, however, only give one suffix).

.PATH If you give sources for this target, *PMake* will take them as directories to search for files it cannot find in the current directory. If you give no sources, it will clear out any directories added to the search path before.

.PATH*suffix* This does a similar thing to **.PATH**, but it does it only for files with the given suffix. The suffix must have been defined already.

.PRECIOUS Gives the **.PRECIOUS** attribute to each source on the dependency line, unless there are no sources, in which case the **.PRECIOUS** attribute is given to every target in the file.

.RECURSIVE
: Applies the **.MAKE** attribute to all its sources. It does nothing if you don't give it any sources.

.SHELL
: Tells *PMake* to use some other shell than the Bourne Shell. The sources for the target are organized as *keyword=value* strings. If a *value* contains white-space, it may be surrounded by double-quotes to make it a single word. The possible sources are:

  path=*path*
  : Tells where the shell actually resides. If you specify this and nothing else, *PMake* will use the last component of the path to find the specification. Use this if you just want to use a different version of the Bourne or C Shell (*PMake* knows how to use the C Shell too).

  name=*name*
  : This is the name by which the shell is to be known. It is a single word and, if no other keywords are specified (other than **path**), it is the name by which *PMake* attempts to find a specification for the it. You can use this if you would just rather use the C Shell than the Bourne Shell (''**.SHELL: name=csh**'' will do it).

  quiet=*echo-off command*
  : The command *PMake* should send to stop the shell from printing its commands. Once echoing is off, it is expected to remain off until explicitly turned on.

  echo=*echo-on command*
  : The command *PMake* should give to turn echoing back on again.

  filter=*printed echo-off command*
  : Many shells will echo the echo-off command when it is given. This keyword tells *PMake* in what format the shell actually prints the echo-off command. Wherever *PMake* sees this string in the shell's output, it will delete it and any following white-space, up to and including the next newline.

echoFlag=*flag to turn echoing on*
: The flag to pass to the shell to turn echoing on at the start. If either this or the next flag begins with a '–', the flags will be passed to the shell as separate arguments. Otherwise, the two will be concatenated.

errFlag=*flag to turn error checking on*
: Flag to give the shell to turn error checking on at the start.

check=*command to turn error checking on*
: The command to make the shell check for errors or to print the command that's about to be executed (%s indicates where the command to print should go), if hasErrCtl is "no".

ignore=*command to turn error checking off*
: The command to turn error checking off or the command to execute a command ignoring any errors. "%s" takes the place of the command.

hasErrCtl=*yes or no*
: This takes a value that is either yes or no, telling how the "check" and "ignore" commands should be used. NOTE: If this is "no", both the check and ignore commands should contain a \n at their end if the shell requires a newline before executing a command.

The strings that follow these keywords may be enclosed in single or double quotes (the quotes will be stripped off) and may contain the usual C backslash characters.

.SILENT
: Applies the **.SILENT** attribute to each of its sources. If there are no sources on the dependency line, then it is as if you gave *PMake* the –s flag.

.SUFFIXES
: This is used to give new file suffixes for *PMake* to handle. Each source is a suffix *PMake* should recognize. If you give a **.SUFFIXES** dependency line with no sources, *PMake* will forget about all the suffixes it knew (this also clobbers the null suffix). For those targets that need to have suffixes defined, this is how you do it.

In addition to these targets, a line of the form

attribute : *sources*

applies the *attribute* to all the targets listed as *sources* except as noted above.

The .EXPORT target is recognized but not implemented on IRIX.

THE POWER OF SUFFIXES

One of the best aspects of both *make*(1) and *PMake* comes from their understanding of how the suffix of a file pertains to its contents and their ability to do things with a file based solely on its suffix. *PMake* also has the ability to find a file based on its suffix, supporting different types of files being in different directories. The former ability derives from the existence of so-called transformation rules while the latter comes from the specification of search paths using the **.PATH** target.

TRANSFORMATION RULES

A special type of dependency, called a transformation rule, consists of a target made of two known suffixes stuck together followed by a shell script to transform a file of one suffix into a file of the other. The first suffix is the suffix of the source file and the second is that of the target file. For example, the target ".c.o," followed by commands, would define a transformation from files with the ".c" suffix to those with the ".o" suffix. A transformation rule has no source files associated with it, though attributes may be given to one in the usual way. These attributes are then applied to any target that is on the "target end" of a transformation rule. The suffixes that are concatenated must be already known to *PMake* in order for their concatenation to be recognized as a transformation, i.e., the suffixes must have been the source for a .SUFFIXES target at some time before the transformation is defined. Many transformations are defined in the system makefile (see **FILES**), which you should examine for more examples as well as to find what is already available. (You should especially note the various variables used to contain flags for the compilers, assemblers, etc., used to transform the files. These variables allow you to customize the transformations to your own needs without having to redefine them.) A transformation rule may be defined more than once, but only the last such definition is remembered by *PMake*. This allows you to redefine the transformations in the system makefile if you wish.

Transformation rules are used only when a target has no commands associated with it, both to find any additional files on which it depends and to attempt to figure out just how to make the target should it end up being out-of-date. When a transformation is found for a target, another of the seven "local" variables mentioned earlier is defined:

.IMPSRC (<)	The name/path of the source from which the target is to be transformed (the ''implied'' source).

For example, given the following makefile:

```
a.out : a.o b.o
        $(CC) $(.ALLSRC)
```

and a directory containing the files a.o, a.c and b.c, *PMake* will look at the list of suffixes and transformations given in the built-in rules and find that the suffixes ''.c'' and ''.o'' are both known and there is a transformation rule defined from one to the other with the command ''$(CC) $(CFLAGS) —c $(.IMPSRC).'' Having found this, it can then check the modification times of both a.c and b.c and execute the command from the transformation rule as necessary in order to update the files a.o and b.o.

PMake, unlike *make*(1) before it, has the ability to apply several transformations to a file even if the intermediate files do not exist. Given a directory containing a .o file and a .v file, and transformations from .v to .w, .w to .c and .c to .o, *PMake* will define a transformation from .v → .o using the three transformation rules you defined. In the event of two paths between the same suffixes, the shortest path will be chosen between the target and the first existing file on the path. So if there were also a transformation from .w files to .o files, *PMake* would use the path .v → .w → .o instead of .v → .w → .c → .o.

Once an existing file is found, *PMake* will continue to look at and record transformations until it comes to a file to which nothing it knows of can be transformed, at which point it will stop looking and use the path it has already found.

What happens if you have a .o file, a .v file and a .r file, all with the same prefix, and transformations from .v → .o and .r → .o? Which transformation will be used? *PMake* uses the order in which the suffixes were given on the **.SUFFIXES** line to decide between transformations: whichever suffix came first, wins. So if the three suffixes were declared

```
.SUFFIXES : .o .v .r
```

the .v → .o transformation would be applied. Similarly, if they were declared as

```
.SUFFIXES : .o .r .v
```

the .r → .o transformation would be used. You should keep this in mind when writing such rules. Note also that because the placing of a suffix on a **.SUFFIXES** line doesn't alter the precedence of previously-defined transformations, it is sometimes necessary to clear the whole lot of them out and start from scratch. This is what the .SUFFIXES-only line, mentioned earlier, will do.

SEARCH PATHS

PMake also supports the notion of multiple directories in a more flexible, easily-used manner than has been available in the past. You can define a list of directories in which to search for any and all files that aren't in the current directory by giving the directories as sources to the .PATH target. The search will only be conducted for those files used only as sources, on the assumption that files used as targets will be created in the current directory.

The line

```
        .PATH : RCS
```

would tell *PMake* to look for any files it is seeking (including ones made up by means of transformation rules) in the RCS directory as well as the current one. Note, however, that this searching is only done if the file is used only as a source in the makefile; the file cannot be created by commands in the makefile.

A search path specific to files with a given suffix can also be specified in much the same way.

```
        .PATH.h :  h  /usr/include
```

causes the search for header files to be conducted in the ''h'' and ''/usr/include'' directories as well as the current one.

When expanding wildcards, these paths are also used. If the pattern has a recognizable suffix, the search path for that suffix is used. Otherwise, the path defined with the regular .PATH target is used.

When a file is found somewhere other than the current directory, its name is replaced by its full pathname in any ''local'' variables.

Two types of suffixes are given special attention when a search path is defined for them. On IRIX, the C compiler lets you specify where to find header files (.h files) by means of –I flags similar to those used by *PMake*. If a search path is given for any suffix used as a source for the .INCLUDES target, the variable $(.INCLUDES) will be set to contain all the directories on the path, in the order given, in a format which can be passed directly to the C compiler. Similarly, one may give directories to search for libraries to the compiler by means of –L flags. Directories on the search path for a suffix which was the source of the .LIBS target will be placed in the $(.LIBS) variable ready to be passed to the compiler.

LIBRARIES AND ARCHIVES

Two other special forms of sources are recognized by *PMake*. Any source that begins with the characters "–l" (lower-case L) or ends in a suffix that is a source for the **.LIBS** target is assumed to be a library, and any source that contains a left parenthesis in it is considered to be a member (or members) of an archive.

Libraries are treated specially mostly in how they appear in the local variables of those targets that depend on them. Since IRIX supports the –L flag when linking, the name of the library (i.e., its "–l" form) is used in all local variables. *PMake* assumes that you will use the $(.LIBS) variable in the appropriate place.

The process of creating a library or archive can be a painful one, what with all the members having to be kept outside the archive as well as inside it in order to keep them from being recreated. *PMake* has been set up, however, to allow you to reference files that are in an archive in a relatively painless manner. The specification of an archive member is written as:

archive(*member* [*member*...])

Both the open and close parenthesis are required and there may be any number of members between them (except 0, that is). Members may also include wildcards characters. When such a source is examined, it is the modification time of the member, as recorded in the archive, that is used to determine its datedness.

If an archive member has no commands associated with it, *PMake* goes through a special process to find commands for it. First, implicit sources are sought using the "member" portion of the specification. So if you have something like "libmalloc.a(malloc.o)" for a target, *PMake* attempts to find sources for the file "malloc.o," even if it doesn't exist. If such sources exist, *PMake* then looks for a transformation rule from the member's suffix to the archive's (in this case from .o → .a) and tacks those commands on as well.

To make these transformations easier to write, three local variables are defined for the target:

.ARCHIVE (%)	The path to the archive file.
.MEMBER (!)	The actual member name (literally the part in parentheses).
.TARGET (@)	The path to the file which will be archived, if it is only a source, or the same as the **.MEMBER** variable if it is also a target.

Using the transformations already in the system makefile, a makefile for a library might look something like this:

```
OBJS = setup.o doit.o transfer.o shutdown.o
.o.a :
          ...
          rm -f $(.MEMBER)

lib.a : lib.a($(OBJS))
          ar cru $(.TARGET) $(.OODATE)
```

Note that the following .o → .a transformation is bad:

```
.o.a :
          ar r $(.ARCHIVE) $(.TARGET)
          ...
          rm -f $(.TARGET)
```

The reason is simple: you should not execute "ar" on the same archive several times at once. Also, it is much slower than archiving all the .o files at the end.

OUTPUT

When creating targets in parallel, several shells are executing at once, each wanting to write its output to the screen. This output must be captured by *PMake* in some way in order to prevent the screen from being filled with garbage even more indecipherable than one can already get from these programs. *PMake* has two ways of doing this, one of which provides for much cleaner output and a clear delineation between the output of different jobs, the other of which provides a more immediate response so one can tell what is really happening. The former is done by notifying the user when the creation of a given target starts, capturing the output, and transferring it to the screen when the job finishes, preceded by an indication as to which job produced the output. The latter is done by catching the output of the shell (and its children) and buffering it until an entire line is received, then printing that line preceded by the name of the job from which the line came. The name of the job is just the target which is being created by it. Since this second method is preferred, it is the one used by default. The first method will be used if the **–P** flag is given to *PMake*.

PARALLELISM

PMake and *smake* attempts to create several targets at once. The degree of useful concurrency depends on the targets, as well as the number of processors on the machine. On IRIX, the default concurrency value for single processor systems is 2. On multi-processor systems that have more than 1 unrestricted processors, the value is 4 (see *mpadmin*(1) to change the number of unrestricted processors). To change the default concurrency, use the **−J** flag. This flag can be set on the command line, inside a makefile with the **.MAKEFLAGS** target or with the **PMAKE** environment variable.

FILES

Makefile or makefile	default input file
/usr/include/make/system.mk	System makefile (the built-in rules)
/usr/include/make/makelib.mk	.USE target for making archives
/usr/include/make/makelint.mk	.USE target for making lint libraries

ENVIRONMENT

PMAKE	Flags PMake should always use when invoked.

SEE ALSO

make(1) for a more complete explanation of the lower-case flags to *PMake*.

AUTHOR

Adam de Boor

NAME

pr – print files

SYNOPSIS

pr [[–**column**] [–wwidth] [–a]] [–eck] [–ick] [–**drtfp**] [+page] [–**n**ck] [–ooffset] [–llength] [–sseparator] [–**h**header] [file ...]

pr [[–**m**] [–wwidth]] [–eck] [–ick] [–**drtfp**] [+page] [–nck] [–ooffset] [–llength] [–sseparator] [–**h**header] file1 file2 ...

DESCRIPTION

pr is used to format and print the contents of a file. If *file* is –, or if no files are specified, *pr* assumes standard input. *pr* prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, the date and time that the file was last modified, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with –**h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the –s option is used, lines are not truncated and columns are separated by the *separator* character.

Either —*column* or –**m** should be used to produce multi-column output. –a should only be used with –*column* and not –**m**.

Command line options are

+page	Begin printing with page numbered *page* (default is 1).
–column	Print *column* columns of output (default is 1). Output appears as if –e and –i are turned on for multi-column output. May not use with –**m**.
–a	Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.
–**m**	Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with –*column*.

−d Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.

−e*ck* Expand input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

−i*ck* In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

−w*width* Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-column* and **−m**). There is no line limit for single column output.

−o*offset* Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

−l*length* Set the length of a page to *length* lines (default is 66). **−l0** is reset to **−l66**. When the value of *length* is 10 or less, **−t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When **−l***length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.

−h *header* Use *header* as the text line of the header to be printed instead of the file name. **−h** is ignored when **−t** is specified or **−l***length* is specified and the value of *length* is 10 or less. (**−h is the only** *pr* option requiring space between the option and argument.)

−p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on files that will not open.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **−t** overrides the **−h** option.

−s*separator*
Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **−w** is specified.

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by ''file list'':

```
pr −3dh "file list" file1 file2
```

Copy **file1** to **file2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr −e9 −t <file1 >file2
```

Print **file1** and **file2** simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
pr −t −n file1 | pr −t −m −n file2 -
```

FILES

/dev/tty* If standard output is directed to one of the special files /dev/tty*, then other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

SEE ALSO

cat(1), pg(1).

NAME

printenv – print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, *printenv* returns exit status 1, else it returns status 0.

SEE ALSO

sh(1), environ(5), csh(1)

NAME

prof — analyze profile data

SYNOPSIS

prof [options] [prog_name [pcsampling_data_file ...]]
prof —pixie [*options*] [*prog_name* [*bbaddrs_file* [*bbcounts_file* ...]]

DESCRIPTION

Prof analyzes one or more data files generated by the MIPS compiler's execution-profiling system and produces a listing. Prof can also combine those data files or produce a feedback file that lets the optimizer take into account the program's runtime behavior during a subsequent compilation. Profiling is a three-step process: first compile the program, then execute it, and finally run *prof* to analyze the data.

The compiler system provides two kinds of profiling:

1. *pc-sampling* interrupts the program periodically, recording the value of the program counter.

2. *basic-block counting* divides the program into blocks delimited by labels, jump instructions, and branch instructions. It counts the number of times each block executes. This provides more detailed (line by line) information than pc-sampling.

Using pc-sampling

To use pc-sampling, compile your program with the option **-p** (strictly speaking, it is sufficient to use this option only when linking the program.) Then run the program, which allocates extra memory to hold the profile data, and (provided the program terminates normally or calls *exit*(2)) records the data in a file at the end of execution.

The environment variable PROFDIR determines the name of the pc-sampling data file and determines whether pc-sampling takes place: if it is not set, the pc-sampling data file is named "mon.out"; if it is set to the empty string, no profiling occurs; if it is set to a non-empty string, the file is named "string/pid.progname," where "pid" is the process id of the executing program and "progname" is the program's name, as it appears in argv[0]. The subdirectory "string" must already exist.

After running your program, use *prof* to analyze the pc-sampling data file.

For example:

```
cc -c myprog.c
cc -p -o myprog myprog.o
myprog                        (generates "mon.out")
prof myprog mon.out
```

When you use *prof* for pc-sampling, the program name defaults to *a.out* and the pc-sampling data file name defaults to *mon.out*; if you specify more than one pc-sampling data file, *prof* reports the sum of the data.

Executables that call *sproc*(2) can obtain pc-sample data for each execution thread in the job. This includes FORTRAN programs that use **−mp** or **−pfa**. In this case, multiple data files are produced. By default, the names are "pid.progname".

For example:

```
f77 -c -mp myprog.f
f77 -p -mp -o myprog myprog.o
myprog              (generates multiple "#####.myprog" files)
```

Using basic-block counting

To use basic-block counting, compile your program without the option −p. Use *pixie*(1) to translate your program into a profiling version and generate a file, whose name ends in ".Addrs", containing block addresses. Then run the profiling version, which (assuming the program terminates normally or calls *exit*(2)) will generate a file, whose name ends in ".Counts", containing block counts. Then use *prof* with the **−pixie** option to analyze the bbaddrs and bbcounts files. Notice that you must tell *prof* the name of your original program, not the name of the profiling version.

For example:

```
cc -c myprog.c
cc -o myprog myprog.o
pixie -o myprog.pixie myprog        (generates "myprog.Addrs")
myprog.pixie                        (generates "myprog.Counts")
prof -pixie myprog myprog.Addrs myprog.Counts
```

When you use *prof* with the **−pixie** option, the program name defaults to *a.out*, the bbaddrs file name defaults to "*program_name*.Addrs", and the bbcounts file name defaults to "*program_name*.Counts". If you specify more than one bbcounts file (never specify more than one bbaddrs file), *prof* reports the sum of the data.

pixie may also be used with programs that call *sproc*(2). In particular, it can be used with multi-processed FORTRAN executables. As with pc-sampling, a separate data file is generated for each execution thread. The data files have the 5 digit process id appended to them.

For example:

```
f77 -c -mp myprog.f
f77 -mp -o myprog myprog.o
pixie -o myprog.pixie myprog        (generates "myprog.Addrs")
myprog.pixie                (generates multiple "myprog.Counts" files)
```

Options to *prof*

For each *prof* option, you need type only enough of the name to distinguish it from the other options (usually the first character is sufficient). Unless otherwise noted, each part of the listing operates only on the set of procedures that results from the combination of the **–exclude** and **–only** options.

If the options you specify would neither produce a listing nor generate a file, *prof* uses **–procedures** plus **–heavy** by default.

–note *comment string*
: If you use this argument, the *"comment string"* appears near the beginning of the listing as a comment.

–pixie Selects pixie mode, as opposed to pc-sampling mode.

–procedures
: Reports time spent per procedure (using data obtained from pc-sampling or basic-block counting; the listing tells which one). For basic-block counting, this option also reports the number of invocations per procedure.

–heavy Reports the most heavily used lines in descending order of use (requires basic-block counting).

–lines Like **–heavy**, but gives the lines in order of occurrence.

–invocations
: For each procedure, reports how many times the procedure was invoked from each of its possible callers (requires basic-block counting). For this listing, the **–exclude** and **–only** options apply to callees, but not to callers.

–zero Prints a list of procedures that were never invoked (requires basic-block counting).

–testcoverage
: Reports all lines that never executed (requires basic-block counting).

–feedback *filename*
: Produces a file with information that the compiler system can use to decide what parts of the program will benefit most from global optimization and what parts will benefit most from in-line

procedure substitution (requires basic-block counting).

—merge *filename*
: Sums the pc-sampling data files (or, in pixie mode, the bbcounts files) and writes the result into a new file with the specified name. The **–only** and **–exclude** options have no affect on the merged data.

—only *procedure_name*
: If you use one or more **–only** options, the profile listing includes only the named procedures, rather than the entire program. If any option uses an uppercase "O" for "Only," *prof* uses only the named procedures, rather than the entire program, as the base upon which it calculates percentages.

—exclude *procedure_name*
: If you use one or more **–exclude** options, the profiler omits the specified procedure from the listing. If any option uses an uppercase "E" for "Exclude," *prof* also omits that procedure from the base upon which it calculates percentages.

—clock *megahertz*
: Alters the appropriate parts of the listing to reflect the clock speed of the CPU. If, for example, the cpu with which the profile data was generated had a 12.5 megahertz clock, then "–clock 12.5" would be appropriate. If you do not specify **–clock** *megahertz* then no calculation of cpu seconds is made from the data.

—quit *n* Truncates the **–procedures** and **–heavy** listings. It can truncate after *n* lines (if *n* is an integer), after the first entry that represents less than *n* percent of the total (if *n* is followed immediately by a "%" character), or after enough entries have been printed to account for *n* percent of the total (if *n* is followed immediately by "cum%"). For example, "–quit 15" truncates each part of the listing after 15 lines of text, "–quit 15%" truncates each part after the first line that represents less than 15 percent of the whole, and "–quit 15cum%" truncates each part after the line that brought the cumulative percentage above 15 percent.

—warnings
: Eliminates warnings about synonymous procedures and alternate entries. These usually appear only on large Fortran programs. Once you have seen the warnings for a particular program you may prefer to suppress them.

FILES

crt1.o	normal startup code
mcrt1.o	startup code for pc-sampling
libprof1.a	library for pc-sampling
mon.out	default pc-sampling data file

SEE ALSO

monitor(3C), profil(2), pixie(1), cc(1), pc(1), f77(1), as(1)

FEATURES

Provided you do not use **–pixie**, *prof* processes "mon.out" files produced by earlier versions of the compiler system using the obsolete **–p2** or **–p3** options.

BUGS

prof does not yet take into account interactions among floating-point instructions.

prof cannot profile programs which have been loaded with shared libraries and will exit with an error message.

prof cannot be run on programs which have been stripped. This includes programs loaded with either the –s or **–x** option.

Multi-processed FORTRAN programs that use pc-sampling must use the default profiling. User level calls to *monitor*(3), *moncontrol*(3), and *monstartup*(3) are not supported.

NAME

prs – print an SCCS file

SYNOPSIS

prs [–**d**[dataspec]] [–**r**[SID]] [–**e**] [–**l**] [–**c**[date-time]] [–**a**] files

DESCRIPTION

prs prints, on the standard output, parts or all of an SCCS file [see *sccsfile*(4)] in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

–d*[dataspec]*	Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.
–r*[SID]*	Used to specify the *SCCS* ID*entification* (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.
–e	Requests information for all deltas created *earlier* than and including the delta designated via the **–r** keyletter or the date given by the **–c** option.
–l	Requests information for all deltas created *later* than and including the delta designated via the **–r** keyletter or the date given by the **–c** option. c date-time The cutoff date-time –**c**[cutoff]] is in the form: YY[MM[DD[HH[MM[SS]]]]]
–c*[date-time]*	Units omitted from the date-time default to their maximum possible values; that is, **–c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date in the form:

"−c77/2/2 9:22:25".

−a Requests printing of information for both removed, i.e., delta type = *R*, [see *rmdel*(1)] and existing, i.e., delta type = *D*, deltas. If the −a keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file [see *sccsfile*(4)] have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.
A tab is specified by \t and carriage return/new-line is specified by \n. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D*˜or˜*R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S

TABLE 1. SCCS Files Data Keywords (continued)

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS:˜:DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS:˜:DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS:˜:DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes*˜or˜*no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes*˜or˜*no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes*˜or˜*no*	S
:J:	Joint edit flag	"	*yes*˜or˜*no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User-defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes*˜or˜*no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y:˜:M:˜:I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt:˜=˜:DT:˜:I:˜:D:˜:T:˜:P:˜:DS:˜:DP:

EXAMPLES

```
prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file
```

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case:*

```
prs s.file
```

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the –a keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), sccsfile(4).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – report process status

SYNOPSIS

ps [options]

DESCRIPTION

ps prints certain information about active processes. Without *options*, information is printed about processes associated with the controlling terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are given in descending order according to volume and range of information provided:

—e	Print information about every process now running.
—d	Print information about all processes except process group leaders.
—a	Print information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
—f	Generate a full listing. (See below for significance of columns in a full listing.)
—l	Generate a long listing. (See below.)
—n *name*	Take argument signifying an alternate system *name* in place of **/unix**.
—t *termlist*	List only process data associated with the terminal given in *termlist*. Terminal identifiers consist of the device's name (e.g., **ttyd1**, **ttyq1**).
—p *proclist*	List only process data whose process ID numbers are given in *proclist*.
—u *uidlist*	List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the **–f** option, which prints the login name.
—g *grplist*	List only process data whose process group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

Under the –f option, *ps* tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the –f option, in square brackets.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (full or long, respectively) that causes the corresponding heading to appear; all means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F (l) Flags (hexadecimal and additive) associated with the process:

01	Process is a system (resident) process.
02	Process is being traced [*ptrace*(2)].
04	Stopped process has been given to parent via *wait*(2).
08	Process is sleeping at a non-interruptible priority.
10	Process is in core.
20	Process user area is in core.
80	Process in stream poll or select.

S	(l)	The state of the process: S Process is sleeping, waiting for a resource. R Process is running. Z Process is terminated and parent not waiting [*wait*(2)]. T Process is stopped [see *ptrace*(2)]. I Process is in intermediate state of creation. X Process is waiting for memory.
UID	(f,l)	The user ID number of the process owner (the login name is printed under the **–f** option).
PID	(all)	The process ID of the process (this datum is necessary in order to kill a process).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
PRI	(l)	The priority of the process (higher numbers mean lower priority). If the process is running with a non-degrading priority, its PRI field is prefixed with a "+" (refer to *schedctl*(2) for information about non-degrading priorities).
NI	(l)	Nice value, used in priority computation.
P	(l)	If the process is running, gives the number of processor on which the process is executing. Contains an asterisk otherwise.
SZ	(l)	Total size (in pages) of the process, including code, data, shared memory, mapped files, shared libraries, and stack. Pages associated with mapped devices are not counted. A page is 4096 bytes.
RSS	(l)	Total resident size (in pages) of process. This includes only those pages of the process that are physically resident in memory. Mapped devices (such as graphics) are not included. Parts of the process that are shared (code, shared libraries, mapped files) have the number of pages pro-rated by the number of processes sharing the page. A page is 4096 bytes.
WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).
STIME	(f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)

TTY	(all)	The controlling terminal for the process (the message, **?**, is printed when there is no controlling terminal).
TIME	(all)	The cumulative execution time for the process.
COMMAND	(all)	The command name (the full command name and its arguments are printed under the **−f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

FILES

/dev	
/dev/sxt/*	
/dev/tty*	
/dev/xt/*	terminal (“tty”) names searcher files
/dev/kmem	kernel virtual memory
/etc/passwd	UID information supplier
/tmp/.ps_data	internal data structure

SEE ALSO

kill(1), nice(1).
getty(1M) in the *System Administrator's Reference Manual.*

WARNING

Things can change while *ps* is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, *ps* checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, *ps* will not find a controlling terminal, so there will be no report.

ps **−ef** may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

NAME

psh – NeWS PostScript shell

SYNOPSIS

psh [files]

DESCRIPTION

psh opens a connection to the NeWS server and transmits the file arguments (or stdin if no files are specified) to it. Any output from NeWS is copied to stdout. The files should be PostScript programs for the NeWS server to execute.

A common use for *psh* is in creating applications written entirely in PostScript. First, type your PostScript program into a file. Then, type as its first line:

```
#! /usr/NeWS/bin/psh
```

If you now make the file executable (with *chmod*) you can invoke it by name from the shell, and UNIX will use */usr/NeWS/bin/psh* to execute it. *psh* will in turn send your program to the NeWS server.

SEE ALSO

sh(1), say(1)
4Sight User's Guide, Section 2, "Programming in NeWS."

NAME

psterm − NeWS terminal emulator

SYNOPSIS

psterm [options] [command]

DESCRIPTION

psterm is a *termcap*-based terminal emulator program for NeWS. When invoked, it reads the */etc/termcap* entry for the terminal named by the **-t** option, or by the **TERM** environment variable, and arranges to emulate the behavior of that terminal. It forks an instance of *command* (or, by default, the program specified by the **SHELL** environment variable, or *csh* if this is undefined), routing keyboard input to the program and displaying its output.

psterm scales its font to make the number of rows and columns specified in the */etc/termcap* entry for the terminal it is emulating fit the size of its window. It also responds to most of the particular escape sequences that *termcap* defines for that terminal.

OPTIONS

−C route */dev/console* messages to this window, if supported by the operating system.

−f Bring up a reasonably-sized terminal in the lower-left corner of the screen (or in the location specified with the −xy option) instead of having the user define its size and location.

−w wait around after the *command* terminates.

−fl *frame label*
Use the specified string for the frame label.

−il *icon label*
Use the specified string for the icon label. The icon label normally defaults to the name of the host on which *psterm* is running.

−li *lines* specifies the height of the window in characters.

−co *columns*
specifies the width of the window in characters.

−xy *x y* specifies the location of the lower left hand corner of the window (in screen pixel coordinates).

−bg causes *psterm* to place itself in the background by disassociating itself from the parent process and the controlling terminal. If *psterm* is invoked with *rsh*(1), this option will cause the rsh command to complete immediately, rather than hang around until *psterm* exits.

—ls causes *psterm* to invoke the shell as a login shell. In addition, any specified *command* will be passed to the shell with a −c option, rather than being invoked directly, so that the shell can establish any environment variables that may be needed by the command. Further, if *psterm* is invoked via *rsh*(1), the host at the other end of the *rsh* socket will be used as the server, unless a NEWSSERVER environment variable is present.

—pm specifies that a *psterm* should enable *page mode*. When page mode is enabled and a command produces more lines of output that can fit on the screen at once, *psterm* will stop scrolling, hide the cursor, and wait until the user types a character before resuming output. When *psterm* is blocked with a screenfull of data, typing a carriage return or space will cause scrolling to proceed by one line or one screenful, respectively; any other character will cause the next screenfull to appear and be passed through as normal input. This mode can also be enabled or disabled interactively, using the *Page Mode* menu item.

SELECTION

Clicking the left mouse button over a character selects that character. Clicking it beyond the end of the line selects the newline at the end of that line. Clicking the middle mouse button over a character when a primary selection does not exist in that window selects that character. Clicking the middle mouse button over a character when a primary selection does exist in that window extends or shrinks the selection to that character.

Note that selections are made by clicking. Mouse tracking is not implemented yet.

The Copy key (F6) copies the *primary* selection to the *shelf*. The Paste key (F8) copies the contents of the *shelf* to the *insertion point*.

If you make a selection while holding down the Copy key, the selection will be a secondary selection. Subsequently letting go of the Copy key copies the *secondary* selection to the *shelf* and deselects the secondary selection.

Making a selection while holding down the Paste key also makes a secondary selection. It pastes the *primary* selection to the location of the secondary selection and deselects the secondary selection.

Copy and Paste of both primary and secondary selections work across separate invocations of *psterm*.

MENU ITEMS

Psterm adds two items to the top of the standard menu associated with the right hand mouse button. These items permit the page mode and automatic margin modes to be turned on and off. Menu items change according to the state of each mode. For example, if page mode is enabled, the menu item will indicate "Page Mode Off".

FILES

/etc/termcap to find the terminal description.

SEE ALSO

4Sight User's Manual, Section 2, "Programming in NeWS."

BUGS

Emulating some terminal types works better than others, largely because there are incomplete */etc/termcap* entries for them.

A large number of *termcap* fields have yet to be implemented.

Page Mode gets easily confused.

NAME

psview – PostScript previewer for NeWS

SYNOPSIS

psview [option(s)] [–] [PostScript-file]

DESCRIPTION

psview puts up a window and runs the user's PostScript code in it. *psview* uses a portion of the window that has the proper aspect ratio for a standard letter-size page in portrait orientation.

If *PostScript-file* is specified, the PostScript code is taken from that file. If no argument is given, or if a '-' is given as the argument, *psview* reads the PostScript program from standard input.

psview works best with files obeying the Adobe *Document Structuring Conventions*. (Such files contain special comments such as *%%Page:*.) When previewing such a file *psview* lets you flip through the pages with page boundaries being determined by locating the %%Page: comments. *psview* provides a slider to move to any page, and a menu to go to the first, previous, next or last page. Clicking the left mouse button goes to the next page. If the document has only a single page, neither the slider nor the page movement menu entries appear.

All commands may be given from the keyboard. Space or '+' goes to the next page. Backspace or '-' goes to the previous page. '0', '<', or ',' go to the first page. '$', '>', or '.' go to the last page. '^C' or 'q' cause *psview* to exit.

If the file doesn't follow the document structuring conventions, it is treated as having a single page. If the PostScript program in that file calls **showpage**, *psview* will pause. Clicking the left mouse or selecting the *Next Page* menu entry will resume output. When the window is redrawn in response to damage or to selecting *Redisplay* from the menu, any pending output is flushed and the program is executed from the beginning of the file.

The options, which may appear in any order so long as they appear before the file, are:

-x1,y1-x2,y2
: Sets the bounding box of the active part of the psview window. The coordinates are given as floating point values in inches.

—B
: Normally psview draws a box to show the edges of the paper. This stops the box being drawn.

−b<color spec>
Sets the background color. The default is white. <color spec> has the syntax [hr]x,y,z where x, y, and z are either floating point numbers between 0 and 1 or integers between 0 and 255. *h* selects the hsb color model which is the default. *r* selects the rgb color model. An individual color part may be set by, for example **-b***g0.5*. Changing the background color may have strange effects especially if the PostScript file uses color.

−c Sets the background to blue and the foreground to white.

−F<scale>
Gives the window a fixed initial size that defaults to the full height of the screen. *Scale*, which defaults to 1.0, applies a uniform scaling to the default initial size.

−f<color spec>
Sets the foreground color. <color spec> is the same as for the background color. Changing the foreground color may have strange effects especially if the PostScript file uses color.

−l Makes psview use a portion of the window that gives the aspect ratio of a landscape oriented 11 x 7 slide.

−S Makes psview use a portion of the window that gives the aspect ratio of a portrait oriented 6.8 x 11 slide.

−s Makes psview use a portion of the window that gives the aspect ratio of a portrait oriented 5.7 x 11 slide.

−v Makes psview verbose.

SEE ALSO

psh(1), say(1).
4Sight User's Guide, Section 2, "Programming in NeWS."
PostScript Language Reference Manual.

TRADEMARK

PostScript is a registered trademark of Adobe Systems Inc.

NOTES

Assumes a syntactically valid PostScript file.

NAME

puzzle – puzzle game for X

SYNOPSIS

puzzle [-option ...]

OPTIONS

–display *display*
: This option specifies the display to use; see *X(1)*.

–geometry *geometry*
: This option specifies the size and position of the puzzle window; see *X(1)*.

–size *WIDTHxHEIGHT*
: This option specifies the size of the puzzle in squares.

–speed *num*
: This option specifies the speed in tiles per second for moving tiles around.

–picture *filename*
: This option specifies an image file containing the picture to use on the tiles. Try "mandrill.cm." This only works on 8-bit pseudo-color screens.

–colormap
: This option indicates that the program should create its own colormap for the picture option.

DESCRIPTION

Puzzle with no arguments plays a 4x4 15-puzzle. The control bar has two boxes in it. Clicking in the left box scrambles the puzzle. Clicking in the right box solves the puzzle. Clicking the middle button anywhere else in the control bar causes puzzle to exit. Clicking in the tiled region moves the empty spot to that location if the region you click in is in the same row or column as the empty slot.

SEE ALSO

X(1)

BUGS

The picture option should work on a wider variety of screens.

COPYRIGHT

Copyright 1988, Don Bennett.

AUTHOR

Don Bennett, HP Labs

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX system administrator.

NAME

rcp – remote file copy

SYNOPSIS

rcp [**–pv**] file1 file2
rcp [**–p**] [**–rv**] file ... directory

DESCRIPTION

Rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form ''remhost:path'', or a local file name (containing no ':' characters, or a '/' before any ':'s). Hostnames may also take the form ''remuser@remhost'' to use the user name *remuser* rather than the current user name on the remote host.

If the **–r** option is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*(2) on the destination host is used. The **–p** option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*. The –v option causes the file names to be printed as it is copied.

If *path* is not a full path name, it is interpreted relative to your login directory on *remhost*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

Rcp does not prompt for passwords; your current local user name must exist on *remhost* and allow remote command execution via *rsh*(1C).

Rcp handles third party copies, where neither source nor target files are on the current machine.

SEE ALSO

cp(1), ftp(1C), rsh(1C), rlogin(1C), hosts(4), rhosts(4)

BUGS

Rcp doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

If you use *csh*(1), *rcp* will not work if your .cshrc file on the remote host unconditionally executes interactive or output-generating commands. Put these commands inside the following conditional block:

```
   if ($?prompt) then

   endif
```

so they won't interfere with *rcp*, *rsh*, and other non-interactive, *rcmd*(3)-based programs.

NAME

rcpDevice – an interactive *transferdevice* for performing rcp within the WorkSpace.

SYNOPSIS

rcpDevice menu
rcpDevice versionOK
rcpDevice import
rcpDevice export

DESCRIPTION

rcpDevice is an interactive tool used for copying files to or from the WorkSpace and a remote machine. It is a type of *transferdevice*, and therefore understands the generic arguments **menu** and **versionOK**.

The argument **import** is used to copy files from a remote machine to the WorkSpace's currently selected directory. **export** is used to copy the files currently selected in the WorkSpace to a remote machine. *rcpDevice* examines the environment variable $SELECTED, which is set to be the current selection by the WorkSpace.

FILES

/etc/transferDevice/

SEE ALSO

transfermanager(1G), transferdevice(4), cpioArchive(1), tarArchive(1), workspace(1G), rcp(1C)
Programming the IRIS WorkSpace

NAME

rcs – change RCS file attributes

SYNOPSIS

rcs [options] file ...

DESCRIPTION

rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For *rcs* to work, the caller's login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the superuser, or the **-i** option is present.

Files ending in ',v' are RCS files, all others are working files. If a working file is given, *rcs* tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co* (1).

–i	creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, *rcs* tries to place it first into the subdirectory ./RCS, and then into the current directory. If the RCS file already exists, an error message is printed.
–a*logins*	appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.
–A*oldfile*	appends the access list of *oldfile* to the access list of the RCS file.
–e[*logins*]	erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.
–c*string*	sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword $Log$ during checkout (see *co*). This is useful for programming languages without multi-line comments. During *rcs -i* or initial *ci*, the comment leader is guessed from the suffix of the working file.
–l[*rev*]	locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with *ci* or *rcs -u* (see below).
–u[*rev*]	unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Somebody else

unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single '.' or control-D.

–L
: sets locking to *strict*. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

–U
: sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should NOT be used for files that are shared. The default (-L or -U) is determined by your system administrator.

–n*name* [:*rev*]
: associates the symbolic name *name* with the branch or revision *rev*. *Rcs* prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is deleted.

–N*name*[:*rev*]
: same as **-n**, except that it overrides a previous assignment of *name*.

–o*range*
: deletes (''outdates'') the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1–rev2* means revisions *rev1* to *rev2* on the same branch, *–rev* means from the beginning of the branch containing *rev* up to and including *rev*, and *rev–* means from revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.

–q
: quiet mode; diagnostics are not printed.

–s*state* [:*rev*]
: sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed. If *rev* is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for *state*. A useful set of states is *Exp* (for experimental), *Stab* (for stable), and *Rel* (for released). By default, *ci* sets the state of a revision to *Exp*.

–t[*txtfile*]
: writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, *rcs* prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the **-i** option is present,

descriptive text is requested even if **-t** is not given. The prompt is suppressed if the std. input is not a terminal.

DIAGNOSTICS

The RCS file name and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

FILES

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. *Rcs* creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, *rcs* always creates a new file. On successful completion, *rcs* deletes the old one and renames the new one. This strategy makes links to RCS files useless.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.1 ; Release Date: 83/04/04.
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

check(1), co(1), ci(1), ident(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4).
Walter F. Tichy, ''Design, Implementation, and Evaluation of a Revision Control System,'' in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rcsdiff – compare RCS revisions

SYNOPSIS

rcsdiff [**−biwt**] [**−cefhn**] [**−q**] [**−r***rev1*] [**−r***rev2*] file ...

DESCRIPTION

Rcsdiff runs *diff*(1) to compare two revisions of each RCS file given. A file name ending in ',v' is an RCS file name, otherwise a working file name. *Rcsdiff* derives the working file name from the RCS file name and vice versa, as explained in *co*(1). Pairs consisting of both an RCS and a working file name may also be specified.

The options **−b**, **−i**, **−w**, **−t**, **−c**, **−e**, **−f**, and **−h**, have the same effect as described in *diff*(1); option **−n** generates an edit script of the format used by RCS. The option **−q** suppresses diagnostic output.

If both *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If *rev1* is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command

```
rcsdiff f.c
```

runs *diff* on the latest revision on the default branch of RCS file f.c,v and the contents of working file f.c.

DIAGNOSTICS

The exit status is 0 if there were no differences during the last comparison, 1 if there were differences, and 2 if there were errors.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number 1.3 89/05/02.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.

SEE ALSO

ci(1), co(1), diff(1), ident(1), rcs(1), rcsintro(1), rcsmerge(1), rlog(1)
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rcsintro – introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, form letters, etc.

The basic user interface is extremely simple. The novice only needs to learn two commands: *ci* and *co*. *ci*, short for ''checkin'', deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. *co*, short for ''checkout'', retrieves revisions from an RCS file.

Functions of RCS

- Storage and retrieval of multiple revisions of text. RCS saves all old revisions in a space efficient way. Changes no longer destroy the original, because the previous revisions remain accessible. Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.
- Maintenance of a complete history of changes. RCS logs all changes automatically. Besides the text of each revision, RCS stores the author, the date and time of checkin, and a log message summarizing the change. The logging makes it easy to find out what happened to a module, without having to compare source listings or having to track down colleagues.
- Resolution of access conflicts. When two or more programmers wish to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.
- Maintenance of a tree of revisions. RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.
- Merging of revisions and resolution of conflicts. Two separate lines of development of a module can be coalesced by merging. If the revisions to be merged affect the same sections of code, RCS alerts the user about the overlapping changes.
- Release and configuration control. Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.

- Automatic identification of each revision with name, revision number, creation time, author, etc. The identification is like a stamp that can be embedded at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.
- Minimization of secondary storage. RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.

Getting Started with RCS

Suppose you have a file f.c that you wish to put under control of RCS. Invoke the checkin command

```
ci f.c
```

This command creates the RCS file f.c,v, stores f.c into it as revision 1.1, and deletes f.c. It also asks you for a description. The description should be a synopsis of the contents of the file. All later checkin commands will ask you for a log entry, which should summarize the changes that you made.

Files ending in ,v are called RCS files ('v' stands for 'versions'), the others are called working files. To get back the working file f.c in the previous example, use the checkout command

```
co f.c
```

This command extracts the latest revision from f.c,v and writes it into f.c. You can now edit f.c and check it back in by invoking

```
ci f.c
```

Ci increments the revision number properly. If *ci* complains with the message

```
ci error: no lock set by <your login>
```

then your system administrator has decided to create all RCS files with the locking attribute set to 'strict'. In this case, you should have locked the revision during the previous checkout. Your last checkout should have been

```
co –l f.c
```

Of course, it is too late now to do the checkout with locking, because you probably modified f.c already, and a second checkout would overwrite your modifications. Instead, invoke

```
rcs –l f.c
```

This command will lock the latest revision for you, unless somebody else got ahead of you already. In this case, you'll have to negotiate with that person.

Locking assures that you, and only you, can check in the next update, and avoids nasty problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, etc. All that locking prevents is a CHECKIN by anybody but the locker.

If your RCS file is private, i.e., if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner of the RCS file need not have a lock for checkin; all others still do. Turning strict locking off and on is done with the commands

```
rcs –U f.c     and     rcs –L f.c
```

If you don't want to clutter your working directory with RCS files, create a subdirectory called RCS in your working directory, and move all your RCS files there. RCS commands will look first into that directory to find needed files. All the commands discussed above will still work, without any modification. (Actually, pairs of RCS and working files can be specified in 3 ways: (a) both are given, (b) only the working file is given, (c) only the RCS file is given. Both RCS and working files may have arbitrary path prefixes; RCS commands pair them up intelligently).

To avoid the deletion of the working file during checkin (in case you want to continue editing), invoke

```
ci –l f.c     or     ci –u f.c
```

These commands check in f.c as usual, but perform an implicit checkout. The first form also locks the checked in revision, the second one doesn't. Thus, these options save you one checkout operation. The first form is useful if locking is strict, the second one if not strict. Both update the identification markers in your working file (see below).

You can give *ci* the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, etc., and you would like to start release 2. The command

```
ci –r2 f.c     or     ci –r2.1 f.c
```

assigns the number 2.1 to the new revision. From then on, *ci* will number the subsequent revisions with 2.2, 2.3, etc. The corresponding *co* commands

```
co –r2 f.c     and     co –r2.1 f.c
```

retrieve the latest revision numbered 2.x and the revision 2.1, respectively. *Co* without a revision number selects the latest revision on the "trunk", i.e., the highest revision with a number consisting of 2 fields. Numbers with more than 2 fields are needed for branches. For example, to start a branch at revision 1.3, invoke

```
ci −r1.3.1 f.c
```

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see *rcsfile*(4).

Automatic Identification

RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker

```
$Header$
```

into your text, for instance inside a comment. RCS will replace this marker with a string of the form

```
$Header: filename revision_number date time author state $
```

With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps the markers up to date automatically. To propagate the markers into your object code, simply put them into literal character strings. In C, this is done as follows:

```
static char rcsid[] = "$Header$";
```

The command *ident* extracts such markers from any file, even object code and dumps. Thus, *ident* lets you find out which revisions of which modules were used in a given program.

You may also find it useful to put the marker $Log$ into your text, inside a comment. This marker accumulates the log messages that are requested during checkin. Thus, you can maintain the complete history of your file directly inside it. There are several additional identification markers; see *co*(1) for details.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907. Copyright © 1982 by Walter F. Tichy.

SEE ALSO

check(1), ci(1), co(1), ident(1), merge(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on*

Software Engineering, IEEE, Tokyo, Sept. 1982.

NAME

rcsmerge – merge RCS revisions

SYNOPSIS

rcsmerge **–r***rev1* [**–r***rev2*] [**–p**] file

DESCRIPTION

rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If **–p** is given, the result is printed to the standard output, otherwise the result overwrites the working file.

A file name ending in ',v' is an RCS file name, otherwise a working file name. *merge* derives the working file name from the RCS file name and vice versa, as explained in *co*(1). A pair consisting of both an RCS and a working file name may also be specified.

rev1 may not be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in the **–j** version of *co*. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose you have released revision 2.8 of f.c. Assume furthermore that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file f.c and execute

```
rcsmerge –p –r2.8 –r3.4 f.c >f.merged.c
```

Then examine f.merged.c. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute *co –j*:

```
ci -r2.8.1.1 f.c
co -r3.4 –j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in f.c.

```
rcsmerge –r2.8 –r2.4 f.c
```

Note the order of the arguments, and that f.c will be overwritten.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 83/01/15.
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci(1), co(1), merge(1), ident(1), rcs(1), rcsdiff(1), rlog(1), rcsfile(4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

rcsmerge does not work for files that contain lines with a single '.'.

NAME

rdist — remote file distribution program

SYNOPSIS

rdist [−nqbRhivwyD] [−f distfile] [−d var=value] [−m host] [name ...]

rdist [−nqbRhivwyD] −c name ... [login@]host[:dest]

DESCRIPTION

Rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. *Rdist* reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is '−', the standard input is used. If no **−f** option is present, the program looks first for 'distfile', then 'Distfile' to use as the input. If no names are specified on the command line, *rdist* will update all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

The **−c** option forces *rdist* to interpret the remaining arguments as a small *distfile*. The equivalent distfile is as follows.

```
( name ... ) -> [login@]host
        install   [dest] ;
```

Other options:

−d Define *var* to have *value*. The **−d** option is used to define or override variable definitions in the *distfile*. *Value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

−m Limit which machines are to be updated. Multiple **−m** arguments can be given to limit updates to a subset of the hosts listed the *distfile*.

−n Print the commands without executing them. This option is useful for debugging *distfile*.

−q Quiet mode. Files that are being modified are normally printed on standard output. The **−q** option suppresses this.

−R Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truly identical copies of directories.

—h Follow symbolic links. Copy the file that the link points to rather than the link itself.

—i Ignore unresolved links. *Rdist* will normally try to maintain the link structure of files being transferred and warn the user if all the links cannot be found.

—v Verify that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

—w Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This will preserve the directory structure of the files being copied instead of flattening the directory structure. For example, renaming a list of files such as (dir1/f1 dir2/f2) to dir3 would create files dir3/dir1/f1 and dir3/dir2/f2 instead of dir3/f1 and dir3/f2.

—y Younger mode. Files are normally updated if their *mtime* and *size* (see *stat*(2)) disagree. The –y option causes *rdist* not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

—b Binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

—D Print debugging messages.

Distfile contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

```
<variable name> '=' <name list>
[ label: ] <source list> '->' <destination list> <command list>
[ label: ] <source list> '::' <time_stamp file> <command list>
```

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The *source list* specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The *destination list* is the list of hosts to which these files are to be copied. Each file in the source list is added to a list of changes if the file is out of date on the host which is being updated (second format) or the file is newer than the time stamp file (third format).

Labels are optional. They are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with '#' and end with a newline.

Variables to be expanded begin with '$' followed by one character or a name enclosed in curly braces (see the examples at the end).

The source and destination lists have the following format:

```
        <name>
or
        '(' <zero or more names separated by white-space> ')'
```

The shell meta-characters '[', ']', '{', '}', '*', and '?' are recognized and expanded (on the local host only) in the same way as *csh*(1). They can be escaped with a backslash. The '~' (tilde) character is also expanded in the same way as *csh* but is expanded separately on the local and destination hosts. When the −w option is used with a file name that begins with '~', everything except the home directory is appended to the destination name. File names which do not begin with '/' or '~' use the destination user's home directory as the root directory for the rest of the file name.

The command list consists of zero or more commands of the following format.

```
        'install'      <options>       opt_dest_name ';'
        'notify'       <name list>     ';'
        'except'       <name list>     ';'
        'except_pat'   <pattern list>  ';'
        'special'      <name list>     string ';'
```

The *install* command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *Opt_dest_name* is an optional parameter to rename files. If no *install* command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name will be created if they do not exist on the remote host. To help prevent disasters, a non-empty directory on a target host will never be replaced with a regular file or a symbolic link. However, under the '−**R**' option a non-empty directory will be removed if the corresponding filename is completely absent on the master host. The *options* are '−**R**', '−**h**', '−**i**', '−**v**', '−**w**', '−**y**', and '−**b**' and have the same semantics as options on the command line except they only apply to the files in the source list. When specifying several options, each one must begin with a hyphen. For example, use **−b −v** instead of **−bv**, which is interpreted as the destination directory. The login name used on the destination host is the same as the local host unless the destination name is of the format ''login@host".

The *notify* command is used to mail the list of files updated (and any errors that may have occurred) to the listed names. If no '@' appears in the name, the destination host is appended to the name (e.g., name1@host, name2@host, ...).

The *except* command is used to update all of the files in the source list except for the files listed in *name list*. This is usually used to copy everything in a directory except certain files.

The *except_pat* command is like the *except* command except that *pattern list* is a list of regular expressions (see *ed*(1) for details). If one of the patterns matches some string within a file name, that file will be ignored. Note that since '\' is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in *pattern list* but not shell file pattern matching characters. To include a '$', it must be escaped with '\'.

The *special* command is used to specify *sh*(1) commands that are to be executed on the remote host after the file in *name list* is updated or installed. If the *name list* is omitted then the shell commands will be executed for every file updated or installed. The shell variable 'FILE' is set to the current filename before executing the commands in *string*. *String* starts and ends with '"' and can cross multiple lines in *distfile*. Multiple commands to the shell should be separated by ';'. Commands are executed in the user's home directory on the host being updated. The *special* command can be used to rebuild private databases, etc. after a program has been updated.

The following is a small example.

```
HOSTS = (sequoia alic@redwood)

FILES = (/d1/project/{src,bin,lib,doc,dbm})

EXLIB = (acct.dir acct.pag)

${FILES} -> ${HOSTS}
                install -R ;
                except /d1/project/dbm/${EXLIB} ;
                special /d1/project/dbm/acct "mkdbm $FILE" ;

srcs:
/d1/project/src -> yosemite
                except_pat ( \\.o\$ /RCS\$ ) ;
```

NAME

regcmp — regular expression compile

SYNOPSIS

regcmp [–] files

DESCRIPTION

The *regcmp* command performs a function similar to *regcmp*(3X) and, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.i. If the – option is used, the output will be placed in *file*.c. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.i files may thus be *included* in C programs, or *file*.c files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name   "([A–Za–z][A–Za–z0–9_]*)$0"

telno  "\({0,1}([2–9][01][1–9])$0\){0,1} *"
       "([2–9][0–9]{2})$1[ –]{0,1}"
       "([0–9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

NAME

relnotes – on-line release notes viewer

SYNOPSIS

relnotes [**–h**]
relnotes <product>
relnotes [**–t**] <product> <chapter> ...

DESCRIPTION

relnotes is an interface to the on-line release notes. It displays the release notes by product and chapter using the *man*(1) command. It can also show which products' release notes are installed and the title of each chapter installed.

With no arguments, *relnotes* shows which products have release notes installed.

–h	shows how to use the command.
product	shows the chapters that are installed for the named product. If no table of contents file can be found, *relnotes* shows you a list of chapter numbers.
product chapter ...	displays the specified chapter(s) from the named product's release notes. The display is done via the *man*(1) command.
-t *product chapter ...*	prints the specified chapter(s) from the named product's release notes. The resulting hardcopy won't be as nicely formatted as the hardcopy of the release notes that you received with the product tape, but will usually be sufficiently legible.

NOTES

The product and chapter can now be specified without using the –p and –c options, so they are no longer supported.

SEE ALSO

man(1)

FILES

/usr/relnotes/<product>/TC	table of contents file for <product>
/usr/relnotes/<product>/ch*.z	release notes for <product>
/tmp/relnotes.$$	temporary file

NAME

resize - utility to set TERMCAP and terminal settings to current window size

SYNOPSIS

resize [-u] [–s [row col]]

DESCRIPTION

Resize prints a shell command for setting the TERM and TERMCAP environment variables to indicate the current size of *xterm* window from which the command is run. For this output to take effect, *resize* must either be evaluated as part of the command line (usually done with a shell alias or function) or else redirected to a file which can then be read in. From the C shell (usually known as */bin/csh*), the following alias could be defined in the user's *.cshrc*:

```
% alias rs 'set noglob; 'eval resize''
```

After resizing the window, the user would type:

```
% rs
```

Users of versions of the Bourne shell (usually known as */bin/sh*) that don't have command functions will need to send the output to a temporary file and the read it back in with the "." command:

```
$ resize >/tmp/out
$ . /tmp/out
```

OPTIONS

The following options may be used with *resize*:

—u This option indicates that Bourne shell commands should be generated even if the user's current shell isn't */bin/sh.*

—c This option indicates that C shell commands should be generated even if the user's current shell isn't */bin/csh.*

—s [*rows columns*]
This option indicates that that Sun console escape sequences will be used instead of the special *xterm* escape code. If *rows* and *columns* are given, *resize* will ask the *xterm* to resize itself. However, the window manager may choose to disallow the change.

FILES

/etc/termcap for the base termcap entry to modify.
~/.cshrc user's alias for the command.

SEE ALSO

csh(1), tset(1), xterm(1)

AUTHORS

Mark Vandevoorde (MIT-Athena), Edward Moy (Berkeley)
Copyright (c) 1984, 1985 by Massachusetts Institute of Technology.
See *X*(1) for a complete copyright notice.

BUGS

The *-u* or *-c* must appear to the left of *-s* if both are specified.

There should be some global notion of display size; termcap and terminfo need to be rethought in the context of window systems. (Fixed in 4.3BSD, and Ultrix-32 1.2)

NAME

Restore – restore the specified file or directory from tape

SYNOPSIS

Restore [*–h hostname*] [*directory name* | *file name*]

DESCRIPTION

The *Restore* command copies the named file or directory from a local or remote backup tape(s) to disk. if no file or directory is specified *Restore* will copy all the files found on the tape to disk.

Files are restored into the current directory if the backup tape contains "." relative path names.

Files on disk are overwritten even if they are more recent than the respective files on tape.

If a tape drive attached to a remote host is used for restore, the name of the remote host needs to be specified with the **–h hostname** option on the command line. For remote restore to successfully work, the user should have a TCP/IP network connection to the remote host and also have "guest" login privileges on that host.

The Restore command expects the backup tape to be in "bru" format.

SEE ALSO

Backup(1), List_tape(1), bru(1).

NAME

rlog – print log messages and other information about RCS files

SYNOPSIS

rlog [options] file ...

DESCRIPTION

rlog prints information about RCS files. Files ending in ',v' are RCS files, all others are working files. If a working file is given, *rlog* tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co*(1).

rlog prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, *rlog* prints complete information. The options below restrict this output.

–L ignores RCS files that have no locks set; convenient in combination with **–R**, **–h**, or **–l**.

–R only prints the name of the RCS file; convenient for translating a working file name into an RCS file name.

–h prints only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.

–t prints the same as **–h**, plus the descriptive text.

–d*dates* prints information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2*, (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co*(1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

–l[*lockers*] prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.

–r*revisions* prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1–rev2* means revisions *rev1* to *rev2* on the same branch, *–rev* means revisions from the beginning of the branch up to and including *rev*, and *rev–* means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

–s*states* prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

–w[*logins*] prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

rlog prints the intersection of the revisions selected with the options **–d**, **–l**, **–s**, **–w**, intersected with the union of the revisions selected by **–b** and **–r**.

EXAMPLES

```
rlog –L –R RCS/*,v
rlog –L –h RCS/*,v
rlog –L –l RCS/*,v
rlog RCS/*,v
```

The first command prints the names of all RCS files in the subdirectory 'RCS' which have locks. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The last command prints complete information.

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.3 ; Release Date: 89/09/12 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rcsfile(4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [**–e***c*] [**–8**] [**–L**] [**–l** username]

DESCRIPTION

Rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost.*

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file .rhosts in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the .rhosts file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The TERM value "iris-ansi" is converted to "iris-ansi-net" when sent to the host. The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. The optional argument **–8** allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than ^S/^Q. The argument –L allows the rlogin session to be run in litout mode. A line of the form "~." disconnects from the remote host, where "~" is the escape character. A line starting with "~!" starts a shell on the IRIS. Similarly, the line "~^Z" (where ^Z, control-Z, is the suspend character) will suspend the rlogin session if you are using *csh*(1). A different escape character may be specified by the –e option. There is no space separating this option flag and the argument character.

SEE ALSO

rsh(1C), hosts(4), rhosts(4)

BUGS

More of the environment should be propagated.

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [–f] [–i] file ...

rm –r [–f] [–i] dirname ... [file ...]

rmdir [–p] [–s] dirname ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with y (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the –f option is in effect.

rmdir removes the named directories, which must be empty.

Three options apply to *rm*:

–f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

–r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the –f option is used, or if the standard input is not a terminal and the –i option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the –f option is used), resulting in an error message.

–i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **–f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

–p This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

–s This option is used to suppress the message printed on standard error when **–p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.
It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm –r .*
```

Both *rm* and *rmdir* return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2) in the *Programmer's Reference Manual*.

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel —rSID files

DESCRIPTION

rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* [see *get*(1)] exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

The –**r** option is used for specifying the *SID* (SCCS IDentification) level of the delta to be removed.

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

x.file	[see *delta*(1)]
z.file	[see *delta*(1)]

SEE ALSO

delta(1), get(1), prs(1), sccsfile(4).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Use *help* (1) for explanations.

NAME

routeprint – route file to printer

SYNOPSIS

routeprint [**–g**] [**–p** printer] [**–t** type] files

DESCRIPTION

routeprint is a utility used by WorkSpace and accessible from the IRIX command line to route files of various types to a set of desired printers. *routeprint* uses file types specified on the command line to look up print conversion rules for each file to be printed. The conversion rules are located in compiled *.ctr* files in */usr/lib/filetype*. The source *.ftr* files can be found in the *local*, *install*, *system*, and *default* subdirectories is */usr/lib/filetype*. If no file types are specified on the command line, *routeprint* looks up the appropriate type for each file. *routeprint* uses the print conversion rules to process the files into a form printable by the target printer.

printer is the name of a printer to which the output may be sent.

type is a file-type name.

files is one or more file names, separated by spaces.

The –g option should be used when routeprint is defined as part of a file typing rule. This option puts error messages in a notifier window (instead of sending them to stdout) and suppresses warnings.

The **–p** or **–t** options may appear multiple times on the command line, and are used in the following way:

–p *printer* is added to the collection of printers on which the output may appear. Each instance of the **–p** option on the command line adds one printer to this collection. If more than one printer is specified, *routeprint* uses the print conversion rules to determine the best printer to use. If no printer names are given via the **–p** flag, the destination printer is the system default printer. Using the **–p** option overrules the system default printer.

–t *type* sets the file-type for the files that follow it on the command line until another type is specified. If no type is given via the **–t** flag, or files appear on the command line before the first **–t**, the files are typed by *routeprint*. (*routeprint* does not currently support the use of multiple file-types.) *routeprint* examines all of the specified files' types. If they are identical, a single print job will be initiated. If the types are varied, *routeprint* generates an error message.

The system default printer is the printer or printer class on which a print job appears if no printer is specified with the **−p** option. The system default printer is normally specified using the Print Manager in the System tool-chest.

USAGE

A typical call from WorkSpace would be from a *.ftr* file type rule entry such as:

```
        PRINT   routeprint   -t   $ARGTYPE   $FIRSTFILE
$RESTFILES
```

A typical call from the command line might look like the following:

```
        routeprint -p myprinter file1 file2 file3
```

JOB ORDERING

The ordering of files handed to *routeprint* determines the ordering of files within the resultant print job. The ordering of files handed to *routeprint* from WorkSpace is constructed in the following manner:

- If an icon is selected individually, its name is appended to the current pending selection list.
- If an area selection is made, each of the icons within that area selection is added to the pending selection list in geographic order, left-to-right, top-to-bottom.

PRINT CONVERSION RULES

The *.ftr* file used by *routeprint* contains both file type rules and print conversion rules.

The following is a typical set of print conversion rules:

```
        CONVERT troff_text postscript
            COST 1
            FILTER psroff -t $file

        CONVERT postscript mylaserprintertype
            COST 1
            FILTER lp -d $CURRENTPRINTER
```

The CONVERT item specifies the file type of the input file followed by the file type of the converted file.

The COST item specifies an arbitrary number between 0 and 100 (inclusive) that represents the image degradation in printing. The higher the COST value, the more *routeprint* will try to avoid printing by that specific conversion method, if it is given a choice.

The FILTER item contains the shell command that performs the conversion.

Given the conversion rules above, the command:

```
routeprint -p mylaserprinter -t troff_text myfile.troff
```

would cause the file mytroff.t to be printed on the printer named "mylaserprinter" via the *psroff* and *lp* commands. Note that more than one conversion rule may be used to actually get the files into a printable form.

FILES

/usr/lib/filetype/local/*.ftr
/usr/lib/filetype/install/*.ftr
/usr/lib/filetype/system/*.ftr
/usr/lib/filetype/default/*.ftr
$HOME/.workspace/print

SEE ALSO

Programming the IRIS WorkSpace

NAME

rpcgen – an RPC protocol compiler

SYNOPSIS

rpcgen [–C *cppcmd*] *infile*
rpcgen **–c|–h|–l|–m** [–C *cppcmd*] [**–o** *outfile*] [*infile*]
rpcgen **–s** *transport* [–C *cppcmd*] [**–o** *outfile*] [*infile*]

DESCRIPTION

rpcgen is a tool that generates C code to implement an RPC protocol. The input to *rpcgen* is a language similar to C known as RPC Language (Remote Procedure Call Language). Information about the syntax of RPC Language is available in the '*rpcgen' Protocol Compiler* chapter in the *Network Communications Guide*.

rpcgen is normally used as in the first synopsis where it takes an input file and generates four output files. If the *infile* is named **proto.x**, then *rpcgen* will generate a header file in **proto.h**, XDR routines in **proto_xdr.c**, server-side stubs in **proto_svc.c**, and client-side stubs in **proto_clnt.c**.

The other synopses shown above are used when one does not want to generate all the output files, but only a particular one. Their usage is described in the USAGE section below.

By default, the C-preprocessor, *cpp*(1), is run on all input files before they are actually interpreted by *rpcgen*, so all the *cpp* directives and C-style comments are legal within an *rpcgen* input file. For each type of output file, *rpcgen* defines a special *cpp* symbol for use by the *rpcgen* programmer:

RPC_HDR	defined when compiling into header files
RPC_XDR	defined when compiling into XDR routines
RPC_SVC	defined when compiling into server-side stubs
RPC_CLNT	defined when compiling into client-side stubs

In addition, *rpcgen* does a little preprocessing of its own. Any line beginning with '%' is passed directly into the output file, uninterpreted by *rpcgen*.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, *rpcgen* will assume that there exists a routine with the name **xdr_** prepended to the name of the undefined type.

OPTIONS

–C *cppcmd*
Run the preprocessor command, *cppcmd*, instead of the default, *cpp*.

−c Compile into XDR routines.

−h Compile into C data-definitions (a header file)

−l Compile into client-side stubs.

−m Compile into server-side stubs, but do not generate a "main" routine. This option is useful for doing callback-routines and for people who need to write their own "main" routine to do initialization.

−o *outfile*
Specify the name of the output file. If none is specified, standard output is used (−c, −**h**, −l and −s modes only).

−s *transport*
Compile into server-side stubs, using the the given transport. The supported transports are **udp** and **tcp**. This option may be invoked more than once so as to compile a server that serves multiple transports.

Make Inference Rules For Compiling XDR Headers

It is possible to set up suffix transformation rules in *make*(1) for compiling XDR routines and header files. The convention is that RPCL protocol files have the extension .x. The *make* rules to do this are:

```
.SUFFIXES: .x
.x.c:
        rpcgen –c $< –o $@

.x.h:
        rpcgen –h $< –o $@
```

SEE ALSO

The RPC chapters in the *Network Communications Guide*.

BUGS

Nesting is not supported. As a work-around, structures can be declared at top-level, and their name used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

NAME

rsh – remote shell

SYNOPSIS

/usr/bsd/rsh host [**–l** username] [**–n**] command

DESCRIPTION

Rsh connects to the specified *host,* and executes the specified *command. Rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the **–l** option. This remote name must be equivalent (in the sense of *rlogin*(1C)) to the originating account; no provision is made for specifying a password with a command.

If you omit *command,* then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1C). In this case, *rsh* understands the additional arguments to *rlogin*.

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
rsh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile,* while

```
rsh otherhost cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile.*

Host names are given in the file */etc/hosts.*

SEE ALSO

rlogin(1C), hosts(4), rhosts(4)

BUGS

If you are using *csh*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of *rsh* to /dev/null using the **–n** option.

You cannot run an interactive command (like *rogue*(6) or *vi*(1)); use *rlogin*(1C).

Job control signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix.

NAME

runon – run a command on a particular cpu

SYNOPSIS

runon n command [arguments]

DESCRIPTION

runon executes *command*, assigning it to run only on cpu *n*, where *n* varies between 0 and the number of processors on the system minus one. (For single-processor systems, then, there is only cpu 0.)

This cpu affinity is inherited across *fork()* and *exec()* system calls. A *sysmp(2)* call can change the cpu affinity. Note that *command* may still run on other processors, briefly, to perform i/o or other hardware-specific actions.

SEE ALSO

mpadmin(1), sysmp(2).

WARNINGS

In the case of the following command

```
runon 1 command1; command2
```

runon applies only to command1. The command

```
runon 2 (command1; command2)
```

is syntactically incorrect.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**–a**] [**–r**] [**–l**] [**–t**] [**–u**]

DESCRIPTION

Ruptime gives a status line like *uptime* for each machine on the local network; these are formed from packets broadcast by each host on the network every three minutes.

Machines for which no status report has been received for 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the **–a** flag is given.

Normally, the listing is sorted by host name. The **–l** , **–t** , and **–u** flags specify sorting by load average, uptime, and number of users, respectively. The **–r** flag reverses the sort order.

FILES

/usr/spool/rwho/whod.*	data files

SEE ALSO

rwho(1C), rwhod(1M)

BUGS

Not all systems keep load statistics that are usable by *rwhod*(1M).

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [–a]

DESCRIPTION

The *rwho* command produces output similar to *who*, but for all machines on the local network. If no report has been received from a machine for 5 minutes then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, then *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of *rwho* unless the –a flag is given.

FILES

/usr/spool/rwho/whod.*	information about other machines

SEE ALSO

ruptime(1C), rwhod(1M)

BUGS

This is unwieldy when the number of machines on the local net is large.

NAME

sact – print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **–e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (i.e., executed a *get* for editing).
Field 4	contains the date that **get –e** was executed.
Field 5	contains the time that **get –e** was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sar – system activity reporter

SYNOPSIS

sar [**–ubdycwaqvmprtghA**] [**–o** file] t [n]

sar [**–ubdycwaqvmprtghA**] [**–s** time] [**–e** time] [**–i** sec] [**–f** file]

DESCRIPTION

sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **–o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file*, either the one specified by **–f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **–s** and **–e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **–i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

–u Report CPU utilization (the default):
%usr, %sys, %intr, %wio, %idle, %sbrk – portion of time running in user mode, running in system mode, processing interrupts, idle with some process waiting for I/O, completely idle or idle with some process waiting because system memory is scarce, respectively. These six percentages add up to 100%. The time that the processor spent in ''idle waiting for I/O'' state is further broken down into the following categories:

%wfs - waiting for file system I/O
%wswp - waiting for swap I/O to complete
%wphy - waiting for physio other than swapping
%wgsw - waiting for graphics context switch to complete
%wfif - waiting while graphics pipe too full

These five numbers add up to 100% of the %wio time.

–b Report buffer activity:
bread/s, bwrit/s – basic blocks transferred between system buffers and disk or other block devices;
lread/s, lwrit/s – basic blocks transferred from system buffers;
%rcache, %wcache – cache hit ratios, i. e., (1–bread/lread) as a percentage;
pread/s, pwrit/s – basic block transfers via raw (physical) device mechanism.

−d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512-byte (basic block) units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

−y Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

−c Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls.

−w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswpout/s – process swapouts
pswch/s – process switches.

−g Report graphics activity:
gcxsw/s - graphics context switches per second
ginpt/s - graphics input driver calls per second
gintr/s - graphics interrupts other than FIFO interrupts per second
fintr/s - FIFO too full interrupts per second
swpbf/s - swap buffers calls per second

−a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc – run queue of processes in memory and runnable;
swpq-sz, %swpocc – swap queue of processes swapped out but ready to run.

−v Report status of process, i-node, file tables and record lock tables:
proc-sz, inod-sz, file-sz, lock-sz – entries/size for each table, evaluated once at sampling point;
ov – overflows that occur between sampling points for each table.

−m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

−p Report paging activities:
vflt/s – address translation page faults (valid page not in memory);
dfill/s – address translation fault on demand fill or demand zero page;
cache/s – address translation fault page reclaimed from page cache;
pgswp/s – address translation fault page reclaimed from swap space;
pgfil/s – address translation fault page reclaimed from file system;
pflt/s – (hardware) protection faults -- including illegal access to page and writes to (software) writeable pages;
cpyw/s – protection fault on shared copy-on-write page;
steal/s – protection fault on unshared writeable page;
rclm/s – pages reclaimed by paging daemon.

Dfill, cache, pgswp, and pgfil are subsets of vflt; cpyw and steal are subsets of pflt.

−t Report translation lookaside buffer (TLB) activities:
tflt/s – user page table or kernel virtual address translation faults: address translation not resident in TLB;
rflt/s – page reference faults (valid page in memory, but hardware valid bit disabled to emulate hardware reference bit);
sync/s – TLBs flushes on all processors;
vmwrp/s – syncs caused by clean (with respect to TLB) kernel virtual memory depletion;
flush/s – single processor TLB flushes;
idwrp/s – flushes because TLB ids have been depleted;
idget/s – new TLB ids issued;
idprg/s – tlb ids purged from process;
vmprg/s – individual TLB entries purged.

−r Report unused memory pages and disk blocks:
freemem – average pages available to user processes;
freeswap – disk blocks available for process swapping.

−h Report system heap statistics:
heapmem – amount of memory currently allocated to system dynamic heap;
allocd – memory in system heap allocated to callers;
overhd – system heap block management overhead;
unused – memory in heap available for allocation;

req/s – number of allocation requests per second;
bk/req – number of blocks searched per request;
breq/s – bytes per second requested of heap;
brnd/s – bytes per second request round-up by heap;
bfree/s – bytes per second freed.

–A Report all data. Equivalent to –**udqbwcayvmprtgh**.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar –o temp 60 10
```

To later review disk and tape activity from that period:

```
sar –d –f temp
```

FILES

/usr/adm/sa/sa*dd*	daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sar(1M) in the *System Administrator's Reference Manual.*

NAME

savemap — saves the current contents of the colormap

SYNOPSIS

savemap file.map [**-r** min max]

DESCRIPTION

savemap saves the current contents of the colormap in a file. The optional arguments allow a specific portion of the color map to be saved.

SEE ALSO

loadmap(1G), makemap(1G)

AUTHOR

Paul Haeberli

NAME

say – execute PostScript

SYNOPSIS

say [*options*] [*strings*]

DESCRIPTION

say connects to the NeWS server and displays the strings provided on the command line in a window. An option is provided to interpret the command line, or the standard input, as a PostScript program to be executed by the server.

say is used to implement some of the NeWS demo programs. This technique allows window applications to be shell scripts.

OPTIONS

–b*string*
: Use *string* as the title for the window.

–c
: Center the text in the window.

–p
: The command line contains a PostScript program rather than simply text strings.

–P
: The standard input contains a PostScript program, which is executed after the PostScript commands on the command line (if any).

–r
: Make the window round.

–s*nn*
: Use *nn* as the point size of the text.

–w
: Wait for the window to be destroyed. The default is for the window to vanish when execution of its PostScript program is finished.

–W
: Do not create a window for the PostScript to be executed in. This can be used to implement operations that do not require a window; for example toggling drag mode in the window manager, or running PostScript code that creates its own window.

–xxx,yyy
: The first *xxx,yyy* pair of numbers sets the X and Y coordinates of the window. If a second *–xxx,yyy* command line option is given, it sets the size of the window.

USAGE

Older programs that use *say* solely to send PostScript to the NeWS server (by specifying the **-P** -w -W " " options to *say*), should be converted to use *psh*(1).

SEE ALSO

psh(1).
4Sight User's Guide, Section 2, "Programming in NeWS."
PostScript Language Reference Manual.

NAME

sc – spread sheet calculator

SYNOPSIS

sc [*file*]

DESCRIPTION

The spread sheet calculator *sc* is based on rectangular tables, in much the same style as Visicalc or Lotus 123. When it is invoked it presents you with an empty table organized as rows and columns of cells. Each cell may have a label string associated with it and an expression. The expression may be a constant or it may compute something based on other entries.

When *sc* is running, the screen is divided into four regions. The top line is for entering commands. The second line is for messages from *sc*. The third line and the first four columns show the row and column numbers. The rest of the screen forms a window looking at the table. The screen has two cursors: a cell cursor (indicated by a '<' on the screen) and a character cursor (indicated by the terminal's hardware cursor). The cell and character cursors are often the same. They will differ when a command is being typed on the top line.

Commands which use the terminal's control key such as ^N will work both when a command is being typed and when in normal mode.

The cursor control commands and the row, column commands can be prefixed by a numeric argument indicating how many times the command is to be executed. "^U" can be used before the number if the number is to be entered while a command is being typed into the command line.

Cursor control commands:

^N	Move the cell cursor to the next row.
^P	Move the cell cursor to the previous row.
^F	Move the cell cursor forward one column.
^B	Move the cell cursor backward one column.
^H	Backspace one character.

h, j, k, l	Alternate cursor controls (left, down, up, right).

Arrow Keys

The terminal's arrow keys provide another alternate set of cell cursor controls if they exist and are supported in the *termcap* entry. Some terminals have arrow keys which conflict with other control key codes. For example, a terminal could send ^H when the back arrow key is depressed. In these cases, the conflicting arrow key performs the same function as the key combination it mimics.

0	Move the cell cursor to column 0 of the current row.
$	Move the cell cursor to the last valid column in the current row.
^	Move the cell cursor to row 0 of the current column.
#	Move the cell cursor to the last valid row in the current column.
g	Go to a cell. The program will prompt for the name of a cell. Enter a cell number such as "a0" or "ae122".

Cell entry and editing commands:

=	Prompts for an expression which will be evaluated dynamically to produce a value for the cell pointed at by the cell cursor. This may be used in conjunction with ^V to make one entries value be dependent on anothers.
"	Enter a label for the current cell.
<	Enter a label that will be flushed left against the left edge of the cell.
>	Enter a label that will be flushed right against the right edge of the cell.

x	Clears the current cell. You may prefix this command with a count of the number of cells on the current row to clear. Cells cleared with this command may be recalled with any of the variations of the pull command.
e	Edit the value associated with the current cell. This is identical to '=' except that the command line starts out containing the old value or expression associated with the cell.
E	Edit the string associated with the current cell. This is the same as either "leftstring", "rightstring", or "label", with the additional fact that the command line starts out with the old string.
m	Mark a cell to be used as the source for the copy command.
c	Copy the last marked cell to the current cell, updating the row and column references.
^T	Toggle cell display. The current cell's contents are displayed in line one when no command being entered or edited. ^T turns the display on or off.

File operations

G	Get a new database from a file.
P	Put the current database into a file.
W	Write a listing of the current database in a form that matches its appearance on the screen. This differs from the "put" command in that "put"s files are intended to be reloaded with "get", while "write" produces a file for people to look at.
T	Write a listing of the current database to a file, but put ":"s between each field. This is useful for tables that will be further formatted by the *tbl* preprocessor of *nroff*.

M Merges the database from the named file into the current database. Values, expressions and names defined in the named file are written into the current file, overwriting the existing entries at those locations.

Row and Column operations. Members of this class of commands can be used on either rows or columns. The second letter of the command is either a column designator (one of the characters c, j, k, ^N, ^p) or a row designator (one of r, l, h, ^B, ^F). Commands which move or copy cells also modify the variable references in affected cell expressions. Variable references may be frozen by using the "fixed" operator.

ar, ac Creates a new row (column) immediately following the current row (column). It is initialized to be a copy of the current one.

dr, dc Delete this row (column).

pr, pc, pm
Pull deleted rows (columns) back into the spread sheet. The last deleted set of cells is put back into the spread sheet at the current location. *pr* inserts enough rows to hold the data. *pc* inserts enough columns to hold the data. *pm* (merge) does not insert rows or columns. It overwrites the cells beginning at the current cursor location.

ir, ic Insert a new row (column) by moving the row (column) containing the cell cursor, and all following, down (right) one. The new position will be empty.

zr, zc Hide ("zap") the current row (column). This keeps a row or column from being displayed but keeps it in the data base.

vr, vc Removes expressions from the affected rows (columns), leaving only the values which were in the cells before the command was executed.

sr, sc Show hidden rows (columns). Type in a range of rows or columns to be revealed. The command default is the first range of rows or columns currently hidden.

f Sets the output format to be used for printing the numbers in each cell in the current column. Type in two numbers which will be the width in characters of a column and the number of digits which will follow the decimal point. Note that this command has only a column version and does have a second letter.

Region Operations: Region commands affect a rectangular region on the screen. All of the commands in this class start with a slash; the second letter of the command indicates which command to do. The program will prompt for needed paramters. Phrases surrounded by square brackets in the prompt are informational only and may be erased with the backspace key.

/x Clear a region. Cells cleared with this command may be recalled via any of the pull row or column commands.

/c Copy a region to the area starting at the current cell.

/f Fill a region with constant values. The start and increment numbers may be positive or negative.

Miscellaneous commands:

q Exit from *sc*. If you were editing a file, and you modified it, then *sc* will ask about saving before exiting. If you aren't editing a file and haven't saved the data you entered, you will get a chance to save the data before you exit.

^C Alternate exit command.

? Types a brief helpful message.

^G or ESC
Abort entry of the current command.

^R or ^L Redraw the screen.

^V	Types, in the command line, the name of the cell referenced by the cell cursor. This is used when typing in expressions which refer to entries in the table.
^E	Types, in the command line, the expression of the cell referenced by the cell cursor.
^A	Types, in the command line, the value of the cell referenced by the cell cursor.

Expressions that are used with the '=' and 'e' commands have a fairly conventional syntax. Terms may be variable names (from the ^V command), parenthesised expressions, negated terms, and constants. Rectangular regions of the screen may be operated upon with '@' functions such as sum (@sum), average (@avg) and product (@prod). Terms may be combined using many binary operators. Their precedences (from highest to lowest) are: ^; *,/; +,-; <,=,>,<=,>=; &; |; ?.

e+e	Addition.
e-e	Subtraction.
e*e	Multiplication.
e/e	Division.
e^e	Exponentiation.
@sum(v:v)	Sum all valid (nonblank) entries in the region whose two corners are defined by the two variable (cell) names given.
@avg(v:v)	Average all valid (nonblank) entries in the region whose two corners are defined by the two variable (cell) names given.

@prod(v:v)	Multiply together all valid (nonblank) entries in the region whose two corners are defined by the two variable (cell) names given.
e?e:e	Conditional: If the first expression is true then the value of the second is returned, otherwise the value of the third is.
<,=,>,<=,>=	Relationals: true iff the indicated relation holds.
&,|	Boolean connectives.
fixed	To make a variable not change automatically when a cell moves, put the word ''fixed'' in front of the reference. I.e. B1*fixed C3

Assorted math functions. Most of these are standard system functions more fully described in *math*(3). All of them operate on floating point numbers (doubles); the trig functions operate with angles in radians.

@exp(expr)	Returns exponential function of <expr>.
@ln(expr)	Returns the natural logarithm of <expr>.
@log(expr)	Returns the base 10 logarithm of <expr>.
@pow(expr1,expr2)	Returns <expr1> raised to the power of <expr2>.
@floor(expr)	Returns returns the largest integer not greater than <expr>.
@ceil(expr)	Returns the smallest integer not less than <expr>.
@hypot(x,y)	Returns SQRT(x*x+y*y), taking precautions against unwarranted overflows.

@fabs(expr)	Returns the absolute value |expr|.
@sin(expr), @cos(expr), @tan(expr)	Return trigonometric functions of radian arguments. The magnitude of the arguments are not checked to assure meaningful results.
@asin(expr)	Returns the arc sin in the range -pi/2 to pi/2
@acos(expr)	Returns the arc cosine in the range 0 to pi.
@atan(expr)	Returns the arc tangent of <expr> in the range -pi/2 to pi/2.
@dtr(expr)	Converts <expr> in degrees to radians.
@rtd(expr)	Converts <expr> in radians to degrees.
pi	A constant quite close to pi.
@max(expr1,expr2)	Returns the largest value of the two expressions.
@min(expr1,expr2)	Returns the smallest value of the two expressions.
@gamma(expr1)	Returns the natural log of the gamma function.

SEE ALSO

bc(1), dc(1).
4Sight User's Guide, Section 2, "Programming in NeWS."

BUGS

At most 200 rows and 40 columns.

NAME

scanner – scan color images

SYNOPSIS

scanner [-r resolution] [-s Xlo,Ylo-Xhi,Yhi] [-t type] [-d device]

DESCRIPTION

scanner reads images from the Sharp FX-450 Color image scanner (and compatible scanners) connected to the VME GPIB card from National Instruments, and also from the Ricoh IS-11 gray-scale and FS-1 color SCSI scanners.

The **-r** option specifies the default resolution, which is also the resolution at which the image is displayed on the screen. The default is 45 DPI (dots per inch) for the Sharp FX-450, 60 DPI for the Ricoh IS-11, and 120 DPI for the Ricoh FS1. The allowable range is from 30 to 300 DPI on the FX-450, 60 to 450 DPI on the IS-11, and 120 to 400 DPI on the FS1. The Howtek scanner is an OEM'ed version of the FX-450 and is also supported.

The **-s** option sets the portion of the scanner bed which is displayed on the screen. The default is to display the entire area (11" x 17" on FX-450, 8.5" x 11" for IS-11 and FS1). This is most useful when scanning small images, since scanning the entire bed can take quite a while, even at 45 DPI. The 4 values are floating point numbers. (Also see re-sizing discussion below).

The -t option can currently be one of *gpib* or *scsi*. If neither is given, the program attempts to use the hardware inventory mechanism to determine which type of scanner is present.

The -d option specifies which device to use. If not given, it defaults to /dev/dev7 for gpib, and to the scsi scanner with the lowest id for scsi (typically /dev/scsi/sc0d7l0).

The right mouse button brings up a sub-menu that allows you to select resolution (of images saved to file only), edge enhancement, brightness, gamma correction, color or b/w, save screen image to a file, previewing the image, a help screen, whether to save as an RGB image or a screen image, and to exit. The defaults are to scan in color, and with sharp edge enhancement, with the initial screen resolution, as an RGB image.

After an image has been scanned and displayed on the screen, the left mouse button is used to select what portion of the image is to be saved to a file. If different resolution, color, or edge enhancement is desired, they must be selected before selecting the area. The section is selected by clicking the left button, then dragging up or down until the desired section is enclosed, then clicking again. If the box dragged out by the mouse corresponds to less than half a inch in either directions on the original picture, no scan will be done. When the button is released, you will be prompted at the bottom of the image for the file to save the image in. Entering any white space character (newline, return, space, or tab) terminates the filename. If no name is entered, the default is scan.rgb or scan.bw depending on whether the image is in color (see **FILES** below).

Once the scan starts, depressing the right mouse button brings up a menu that allows you to cancel the scan (if canceled during a save to file, the image file is removed). Sometimes it may take a while for the program to respond when it is reading data from the scanner. The rectangle marking the area being saved, and the name of the image file remain in the window until the scan is done. The portion of the image being scanned is indicated by a red line within the rectangle.

Resizing the window changes the portion of the image that will be scanned. In general, changing the sizes affects only the top right corner. The existing image, if any continues to be displayed. The exception occurs when the X or Y dimension is increased past the maximum, and the current origin is not at 0. In this case the lower left corner is moved towards the origin, and no image is displayed until the next scan command.

It is possible to select resolution based on either DPI, or to fit the display screen. When the image is chosen to fit the screen, you can also choose to have the aspect ratio match the screen; otherwise the larger dimension is scaled to match the screen size.

NOTES

If a colormap other than that created by the *makemap* program is used, both black and white and color images may not appear correctly. Color images displayed on systems with only 8 bitplanes may not match the original as well as on systems with 24 bitplanes, but the saved file will look 'correct' when displayed with *ipaste* on systems with more than 8 bitplanes. It is also possible to convert them to .im8 (NeWS) images with the *tonews* program and display them correctly even on 8 bitplane systems with the *showimage* program.

Color images at 150 DPI typically take 2-5 Mbytes of disk space, depending on the complexity and area being saved. At 300 DPI, an 8" x 10" color photo can create an image file of more than 8 Mbytes.

FILES

If no file name is given when saving an image, the default is scan.bw or scan.rgb as appropriate.

/dev/dev7 is the default GPIB device opened by *scanner*; the switches on the scanner must be set to 7 to match this. /dev/scsi/sc0d7l0 is the default for SCSI scanners.

SEE ALSO

showimage(1), *tonews*(1), *ipaste*(1), *gpib*(7m), *ds*(7m)

NAME

sccsdiff - compare two versions of an SCCS file

SYNOPSIS

sccsdiff **-r**SID1 **-r**SID2 [**-p**] [-sn] files

DESCRIPTION

sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

—r*SID?*	*SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.
—p	pipe output for each file through *pr*(1).
—s*n*	*n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

get(1).
bdiff(1), help(1), pr(1) in the *User's Reference Manual.*

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.
Use *help*(1) for explanations.

NAME

script – make typescript of terminal session

SYNOPSIS

script [**–a**] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the –a option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

BUGS

Script places **everything** in the log file. This is not what the naive user expects.

NAME

sdiff — side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

The following options exist:

–w *n*	Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
–l	Only print the left side of any lines that are identical.
–s	Do not print identical lines.
–o *output*	Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column

e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** script] [**–f** sfile] [files]

DESCRIPTION

sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **–e** option and no **–f** options, the flag **–e** may be omitted. The **–n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **—n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period **.** matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**	
text	Append. Place *text* on the output before reading the next input line.
(2) **b** *label*	Branch to the **:** command bearing the *label.* If *label* is empty, branch to the end of the script.
(2) **c**	
text	Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
(2) **d**	Delete the pattern space. Start the next cycle.
(2) **D**	Delete the initial segment of the pattern space through the first new-line. Start the next cycle.
(2) **g**	Replace the contents of the pattern space by the contents of the hold space.
(2) **G**	Append the contents of the hold space to the pattern space.
(2) **h**	Replace the contents of the hold space by the contents of the pattern space.
(2) **H**	Append the contents of the pattern space to the hold space.
(1) **i**	
text	Insert. Place *text* on the standard output.
(2) **l**	List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in octal notation and long lines are folded.
(2) **n**	Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
(2) **N**	Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
(2) **p**	Print. Copy the pattern space to the standard output.

(2) **P**	Copy the initial segment of the pattern space through the first new-line to the standard output.
(1) **q**	Quit. Branch to the end of the script. Do not start a new cycle.
(2) **r** *rfile*	Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of **/**. For a fuller description see *ed*(1). *Flags* is zero or more of:

n	n= 1 - 512. Substitute for just the n th occurrence of the *regular expression.*
g	Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
p	Print the pattern space if a replacement was made.
w *wfile*	Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label*	Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.
(2) **w** *wfile*	Write. Append the pattern space to *wfile*.
(2) **x**	Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function*

Don't. Apply the *function* (or group, if *function* is **{**) only to lines *not* selected by the address(es).

(0) **:** *label*	This command does nothing; it bears a *label* for **b** and **t** commands to branch to.
(1) **=**	Place the current line number on the standard output as a line.
(2) **{**	Execute the following commands through a matching **}** only when the pattern space is selected.
(0)	An empty command is ignored.
(0) **#**	If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

setmon – set the default monitor video output format

SYNOPSIS

setmon [options] format

DESCRIPTION

setmon sets the default video output format (VOF) for the specified monitor, providing functionality similiar to the *setenv monitor* command present within the PROM Monitor.

Command line options are:

—n	Specifies that *format* should not be the default monitor, just the current monitor type.
—g	Enables genlock (external video clock).
—s*syncselect*	Used to specify the source of the sync signal. *syncselect* is any combination of "r", "g", "b", and "a" to represent the sync signal on the same combination of the red, green, blue, and alpha video cables. If *syncselect* is "n", the sync signal will be generated on the sync cable. If *syncselect* is not specified, the sync signal will default to the green cable.
—v	Enable verbose mode. On systems supporting custom video output formats, statistics related to downloaded file size and compression are printed.
format	Specifies the desired video output format. There are five supported video output formats. They are **30HZ, 60HZ, NTSC,** and **PAL** (also used to select *SECAM*). On systems that support custom video output formats, a user-defined format may be selected by first placing the file containing the VOF into the appropriate */etc/gl/ucode/vof* subdirectory as user*n*.u, where *n* is within the range of 0 to 9. A given user-defined format may then be selected by specifying the *format* argument as user*n*.

EXAMPLES

```
setmon -sg 30HZ
```

sets the video output format to 30HZ, sync-on-green.

setmon -s g 60HZ

sets the video output format to 60HZ, sync-on-green.

setmon -g -s rgb user4

sets the video output format to the VOF contained in the file user4.u found in the appropriate */etc/gl/ucode/vof* subdirectory. The format is genlocked and with sync-on-red/green/blue.

FILES

/etc/gl/ucode/vof/*/*.u

SEE ALSO

setmonitor(3g)

NAME

setnewshost – generate a string for the NEWSSERVER environment variable

SYNOPSIS

setnewshost *hostname*

DESCRIPTION

setnewshost generates and prints the proper value of the NEWSSERVER environment variable for the given *hostname*. If NEWSSERVER is set then NeWS clients will attempt to connect to the server it points to rather than the local host.

The format of the NEWSSERVER environment variable is as follows:

decimal-address **.** *port#* ; *hostname*

For example, if the host called ''paper'' has address 192.98.34.118, the NEWSSERVER variable should be set to ''3227656822.2000;paper'' so that NeWS clients will connect to the NeWS server on ''paper''. *setnewshost* simply calculates this string and sends it to standard output. This is not its most convenient form, however. C-shell users can define the following alias:

```
alias snh 'setenv NEWSSERVER `setnewshost \!*`'
```

and System V Bourne Shell users can define the following function:

```
snh () {
   NEWSSERVER=`setnewshost $*`
   export NEWSSERVER
}
```

Both forms let you simply type '**snh** *hostname*' to set the NEWSSERVER environment variable automatically.

SEE ALSO

psh(1).
4Sight User's Guide, Section 2, "Programming in NeWS."

BUGS

The host table entry must have exactly the following format: **a.b.c.d**<*tab*>**hostname**.

If you use the **snh** alias or shell function, and the hostname you give is unknown, or you give too many or too few arguments, the NEWSSERVER variable will be trashed.

NAME

setup – initialize system for first user

SYNOPSIS

setup

DESCRIPTION

The *setup* command, which is also accessible as a login by the same name, allows the first user to be established as the "owner" of the machine.

The user is permitted to add the first logins to the system, usually starting with his or her own.

The user can then protect the system from unauthorized modification of the machine configuration and software by giving passwords to the administrative and maintenance functions. Normally, the first user of the machine enters this command through the setup login, which initially has no password, and then gives passwords to the various functions in the system. Any that the user leaves without password protection can be exercised by anyone.

The user can then give passwords to system logins such as "root", "bin", etc. (*provided they do not already have passwords*). Once given a password, each login can only be changed by that login or "root".

The user can then set the date, time and time zone of the machine.

The user can then set the node name of the machine.

SEE ALSO

passwd(1).

DIAGNOSTICS

The *passwd*(1) command complains if the password provided does not meet its standards.

WARNING

If the setup login is not under password control, anyone can put passwords on the other functions.

NAME

sh, rsh – shell, the standard/restricted command programming language

SYNOPSIS

/bin/sh [**–acefhiknrstuvx**] [args]
/bin/rsh [**–acefhiknrstuvx**] [args]

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. *rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *****, **@**, **#**, **?**, **–**, **$**, and **!**.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see "File Name Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is executed in the current (that is, parent) shell.

name () {*list*;}

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (\`\`) and the standard output from these commands may be used as all or part of a word. Trailing new-lines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (\`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...\` ...\` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (\new-line), both the backslash and the new-line are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, \`, ", new-line, and $ are left intact when the command string is read.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:−*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
: If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message ''parameter null or not set'' is printed.

${*parameter*:+*word*}
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:−`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
−	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.

CDPATH
: The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (**:**) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "$ ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL When the shell is invoked, it scans the environment (see "Environment" below) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are *not* passed on as arguments to the invoked command. Note that parameter and command substitution occurs before *word* or *digit* is used.

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[−]word After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, − is appended to <<:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),

2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and

3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,

2) (escaped) **\new-line** is ignored, and

3) \ must be used to quote the characters \, $, and `.

The resulting document becomes the standard input.

<&digit Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**.

<&− The standard input is closed. Similarly for the standard output using **>&−**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under "Commands," if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by & the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Before a command is executed, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (′′ or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **\new-line** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (′′), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "′").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If $* is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("$1 $2 ..."); however, if $@ is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("$1" "$2" ...). \ quotes the characters \, ‵, ", and $. The pair **\new-line** is removed before parameter and command substitution. If a backslash precedes characters other than \, ‵, ", $, and new-line, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set** -a). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                          and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set –k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/usr/sbin:/usr/bsd:/bin:/usr/bin:/usr/bin/X11** (specifying the current directory, **/usr/sbin**, **/usr/bsd**, **/bin**, **/usr/bin**, and **/usr/bin/X11**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

echo [*arg* ...]
Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts Use in shell scripts to support command syntax standards (see *intro*(1)); it parses positional parameters and checks for legal options. See *getopts*(1) for usage and description.

hash [**–r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

limit [**–h**] [*resource* [*maximim-use*]]
Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given. If the –h flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits, but a user may lower or raise the current limits within the legal range.

Resources controllable currently include *cputime*, the maximum number of cpu-seconds to be used by each process, *filesize*, the largest single file which can be created, *datasize*, the maximum growth of the data+stack region via *sbrk*(2) beyond the end of the program text, *stacksize*, the maximum size of the automatically-extended stack region, *coredumpsize*, the size of the largest core dump that will be created, and *memoryuse*, the maximum amount of physical memory a process may have allocated to it at a given time.

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is 'k' or 'kilobytes' (1024 bytes); a scale factor of 'm' or 'megabytes' may also be used. For *cputime* the default scaling is 'seconds', while 'm' for minutes or 'h' for hours, or a time of the form 'mm:ss' giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

newgrp [*arg* ...]
Equivalent to exec **newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using \new-line. Characters other than new-line can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**——aefhkntuvx** [*arg* ...]]

- **–a** Mark variables which are modified or created for export.
- **–e** Exit immediately if a command exits with a non-zero exit status.
- **–f** Disable file name generation
- **–h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- **–k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **–n** Read commands but do not execute them.
- **–t** Exit after reading and executing one command.
- **–u** Treat unset variables as an error when substituting.
- **–v** Print shell input lines as they are read.
- **–x** Print commands and their arguments as they are executed.
- **—** Do not change any of the flags; useful in setting **$1** to **–**.

Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]

The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test

Evaluate conditional expressions. See *test*(1) for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
: For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
: Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own ulimit, but only a super-user (see *su*(1M)) can raise a ulimit.

umask [*nnn*]
: The user file-creation mask is set to *nnn* (see *umask*(1)). If *nnn* is omitted, the current value of the mask is printed.

unlimit [**–h**] [*resource*]
: Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. If **–h** is given, the corresponding hard limits are removed. Only the super-user may do this.

unset [*name* ...]
: For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
: Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is –, commands are initially read from */etc/profile* and from *$HOME/.profile*, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*. The flags below are interpreted by the shell on invocation only; Note that unless the **–c** or **–s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string*	If the −c flag is present commands are read from *string*.
−s	If the −s flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.
−i	If the −i flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that wait is interruptible). In all cases, QUIT is ignored by the shell.
−r	If the −r flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

rsh Only

rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of $PATH,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after *.profile* is interpreted.

A restricted shell can be invoked in one of the following ways: (1) *rsh* is the file name part of the last entry in the */etc/passwd* file (see *passwd*(4)); (2) the environment variable **SHELL** exists and *rsh* is the file name part of its value; (3) the shell is invoked and *rsh* is the file name part of argument 0; (4) the shell is invoke with the **−r** option.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the *.profile* (see *profile*(4)) has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), dup(2), echo(1), env(1), exec(2), fork(2), getopts(1), getrlimit(2), intro(1), login(1), newgrp(1), pipe(2), profile(4), pwd(1), signal(2), test(1), ulimit(2), umask(1), wait(1) in the *Programmer's Reference Manual.*

CAVEATS

Words used for filenames in input/output redirection are not interpreted for filename generation (see "File Name Generation," above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For *wait n*, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

showsnf - print contents of an SNF file

SYNOPSIS

showsnf [-v] [-g] [-m] [-M] [-l] [-L] [-p#] [-u#]

DESCRIPTION

The *showsnf* utility displays the contents of font files in the Server Natural Format produced by *bsdtosnf*. It is usually only to verify that a font file hasn't been corrupted or to convert the individual glyphs into arrays of characters for proofreading or for conversion to some other format.

OPTIONS

−v	This option indicates that character bearings and sizes should be printed.
−g	This option indicates that character glyph bitmaps should be printed.
−m	This option indicates that the bit order of the font is MSBFirst (most significant bit first).
−l	This option indicates that the bit order of the font is LSBFirst (least significant bit first).
−M	This option indicates that the byte order of the font is MSBFirst (most significant byte first).
−L	This option indicates that the byte order of the font is LSBFirst (least significant byte first).
−p#	This option specifies the glyph padding of the font (# is a number).
−u#	This option specifies the scanline unit of the font (# is a number).

SEE ALSO

X(1), Xserver(1), bdftosnf(1)

BUGS

There is no way to just print out a single glyph.

COPYRIGHT

NAME

size – print the section sizes of an object file

SYNOPSIS

size [**–o –d –x –A –B –V**] files

DESCRIPTION

The *size* command prints information about the *.text*, *.init*, *.rdata*, *.data*, *.sdata*, *.sbss* and *.bss* sections of each *file*. The file can be an object or an archive.

Size supports the following options:

—d Print the sizes in decimal.

—o Print the sizes in octal.

—x Print the sizes in hexadecimal.

—A Use AT&T System V style output. This style is more verbose than –B; each section are printed with both size and physical and virtual addresses. –A is the default output style.

—B Use Berkeley (4.3BSD) style output. This style prints size information for each section, regardless of whether the file exists, and prints the total in both hexadecimal and decimal. The *text* and *data* section sizes reported are rounded to page sizes.

—V Print the version of *size*.

SEE ALSO

a.out(4)

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C) in the *Programmer's Reference Manual.*

NAME

sm – a session manager for x

SYNOPSIS

sm [-ignore] [-print] [-debug] [-display hostname:dpy] [normal Xaw options]

DESCRIPTION

sm is a convenient way to save the state of your "desktop" and restore it.

The state is saved in a file called *sm/State* which is located in your home directory. *Sm* executes each of the commands in this file at startup time. If you don't have *sm/State* in your home directory, *sm* first tries to use the file */usr/lib/X11/sm/DefaultState.* If that file doesn't exist then, *sm* executes "xterm –C".

OPTIONS

–ignore
Ignore any existing sm/State file.

–print Just print out that current state.

–debug Turn on debugging.

STARTING AND QUITING

sm is normally started by *xinit* or *xdm after running xrdb* and starting an ICCCM compliant window manager(such as X11R4's twm).

If *sm* was started by *xinit* or *xdm*, quitting *sm* shutdowns the X Window System server. *sm* has a small window with a button that looks like a stop sign. When you click on the shutdown button a confirmation box pops up; if you click on the ''Save State'' button then *sm* will save the state of the system.

OPTIONS

sm takes all of the standard toolkit options. In addition, you may ignore the startup file with -ignore or you can write the current configure to stdout using -print.

X DEFAULTS

The available names and classes for the widgets used are:

NAME	CLASS
print	Boolean
ignore	Boolean
stateFile	String
defaultState	String
shell	String
defaultcommand	String
geometryString	String
iconicString	String
geometryString	String
iconPixmap	Pixmap
geometry	Geometry
homeForm	Form
homeForm.quitButton	Form.CommandButton
confirmForm.confirmLabel	Form.Label
confirmForm.yesButton	Form.Label
confirmForm.noButton.label:	Form.label
confirmForm.checkBoxLabel.label:	Form.label

These can be used to set fonts, colors, etc. tailored to the user's needs. As a color example:

```
sm*quitButton.background:     mauve
sm*yesButton.background:      peach
sm*noButton.background:       plum
sm.geometry:                  -1-100
sm*homeForm.font:             vbee-36
sm*confirmForm.font:          vbee-36
```

FILES

/usr/lib/X11/sm/DefaultState ~/sm/State

BUGS

Sm doesn't iconic windows or window gravity.

AUTHOR

Mike Wexler

mikew@fx.com

NAME

sort − sort and/or merge files

SYNOPSIS

sort [−cmu] [−ooutput] [−ykmem] [−zrecsz] [−dfiMnr] [−btx]
[+pos1 [−pos2]] [files]

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if − is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

−c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

−m Merge only, the input files are already sorted.

−u Unique: suppress all but one in each set of lines having equal keys.

−o*output*
The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **−o** and *output.*

−y*kmem*
The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, −y0 is guaranteed to start with minimum memory. By convention, −y (with no argument) starts with maximum memory.

−z*recsz*
The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the −c or −m options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

–d ''Dictionary'' order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

–f Fold lower-case letters into upper case.

–i Ignore non-printable characters.

–M Compare as months. The first three non-blank characters of the field are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **–M** option implies the **–b** option (see below).

–n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **–n** option implies the **–b** option (see below). Note that the **–b** option is only effective when restricted sort key specifications are in effect.

–r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 –pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *–pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

–b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **–b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *–pos2* argument (see below).

–t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by +*m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; +*m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by –*m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; –*m*.1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 –2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort –r –o outfile +1.0 –1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort –r +1.0b –1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort –t: +2n –3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **–um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort –um +2 –3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the **−c** option. When the last line of an input file is missing a **new-line** character, *sort* appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

NAME

spell, spellin, spellout – find spelling errors

SYNOPSIS

spell [**–v**] [**–b**] [**–x**] [**–d** hlist] [**–s** hstop] [**–h** spellhist] [file] ...

spellin [list]

spellout [**–d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff, tbl* and *eqn*(1) constructions.

Under the **–v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **–b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **–x** option, every plausible stem is printed with '=' for each word.

The spelling list is based on many sources. While it is more haphazard than an ordinary dictionary, it is also more effective with proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

The auxiliary files used for the spelling list, stop list, and history file may be specified by arguments following the **–d**, –s, and –**h** options. The default files are indicated below. Copies of all output may be accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy–y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell*. Both expect a set of words, one per line, from the standard input. *Spellin* combines the words from the standard input and the preexisting *list* file and places a new list on the standard output. If no *list* file is specified, the new list is created from scratch. *Spellout* looks up each word from the standard input and prints on the standard output those that are missing from (or present on, with option –d) the hashed *list* file. For example, to verify that *hookey* is not on the default spelling list, add it to your own private list, and then use it with *spell*,

```
echo hookey | spellout /usr/lib/dict/hlista
echo hookey | spellin /usr/lib/dict/hlista > myhlist
spell −d myhlist huckfinn
```

FILES

/usr/lib/dict/hlista	hashed American spelling list, default for **−d**
/usr/lib/dict/hlistb	hashed British spelling list, default for **−d**
/usr/lib/dict/hstop	hashed stop list, default for **−s**
/usr/lib/dict/words	the dictionary
/dev/null	history file, default for **−h**
/tmp/spell.$$*	temporary files
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1), tee(1), sed(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.
British spelling was done by an American.

NAME

split – split a file into pieces

SYNOPSIS

split [*–n*] [file [name]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if **–** is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

stdump – dump a file of intermediate-code symbolic information

SYNOPSIS

stdump [–a] [–b] [–c] [–e] [–g] [–h] [–i] [–j] [-n number] *file*

DESCRIPTION

The compilers on the IRIS-4D series generate a common intermediate language which is separated into binary instructions and symbolic information, each constituting a separate file. Use *stdump* to dump a file containing symbolic information which was generated by the IRIS-4D compilers. *Stdump* writes to the standard output. The organization of the output is simple but the details of the output are complicated. The detailed output will not be defined here. See the "Assembly Language Programmer's Guide" for additional information on the symbol table. The output of *stdump* is subject to change and one should not rely on its remaining the same from release to release.

The organization of the output is: for each source file represented in *file* there may be auxiliary-symbols, local-symbols, a file-indirect-table, optimization-entries, procedures, and line-numbers; there is only one externals-table and one dense-number-table in *file*.

By default, *stdump* prints all information about all sections of the symbol table. The options (described below) restrict the output to the selected sections.

The *file* may be an object file (such as produced by `cc -c`) or an executable file (such as produced by *ld(1)* or *cc*(1)) or a symbolic information file (which may be produced as described below).

Normally, symbolic information files (and instruction files) are placed in /tmp and removed after each compilation. Use the –K option to *cc*, *f77*(1), or *pc*(1) to force the compiler to preserve these files in the target directory. When this switch is used, the compilation of **file.***x* (where *x* is c for C, f for FORTRAN, or p for Pascal) will produce the intermediate files **file.***B* (instructions) and **file.***T* (symbolic information). The latter file (**file.***T*) may be used as input to *stdump*.

The following options are recognized:

–a	Print the dense number table. This section is empty for object and executable files.
–b	Print the externals table.
–c	Print the local symbols. The source file name is printed here.

−e Print the procedures table. This part of the table gives the per-function information, such as the register-save-mask and the stack frame size.

−g Print the auxiliaries table. This table has encoded in it in a complex way the actual data types of all the data in the symbols. The local-symbol and externals table sections show this data expanded into semi-readable text.

−h Print the line table. One source line number per 32-bits of executable code.

−i Print the File Indirect Table.

−j Print optimization entries. This section is empty for object and executable files.

−n number
Print information about only the source file whose number is given. Files are numbered sequentially starting with zero.

FILES

/tmp/ctmsta*xxxxx*	default name of symbolic information file for process id *xxxxx*
file.T	name of symbolic information file created by the −K option
/usr/bin/stdump	intermediate language symbolic information dump program

SEE ALSO

cc(1), f77(1), pc(1), *Assembly Language Programmer's Guide*

NAME

strings – find the printable strings in an object, or other binary file

SYNOPSIS

strings [**–**] [**–o**] [*–number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. Unless the **–** flag is given, *strings* only looks in the initialized data space of object files. If the **–o** flag is given, then each string is preceded by its offset in the file (in decimal). If the *–number* flag is given, then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

EXAMPLE

```
strings obj1
```

will locate the ASCII-character strings in the object file "obj1".

SEE ALSO

od(1).

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the –s option of *ld*.

FILES

___strip*xxxxx*	temporary file where *xxxxx* is the process id

SEE ALSO

ld(1)

NAME

stty – set the options for a terminal

SYNOPSIS

stty [–a] [**–g**] [options]

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding CTRL character (e.g., ‘‘^**H**’’ is **CTRL-H** ; in this case, recall that **CTRL-H** is the same as the ‘‘backspace’’ key.) The sequence ‘‘^´’’ means that an option has a null value. For example, normally **stty –a** will report that the value of **swtch** is ‘‘^´’’; however, if *csh*(1) is used, **swtch** will have the value ‘‘^**Z**’’.

–a	reports all of the option settings;
–g	reports current settings in a form that can be used as an argument to another *stty* command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (–parenb)	enable (disable) parity generation and detection.
parodd (–parodd)	select odd (even) parity.
cs5 cs6 cs7 cs8	select character size (see *termio*(7)).
0	hang up phone line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
hupcl (–hupcl)	hang up (do not hang up) serial-line connections on last close.
hup (–hup)	same as **hupcl (–hupcl)**.
cstopb (–cstopb)	use two (one) stop bits per character.
cread (–cread)	enable (disable) the receiver.
clocal (–clocal)	assume a line without (with) modem control.
loblk (–loblk)	block (do not block) output from a background job.
tostop (–tostop)	block (do not block) output from a background job (same as **loblk**).

Input Modes

ignbrk (-ignbrk)	ignore (do not ignore) break on input.
brkint (-brkint)	signal (do not signal) INTR on break.
ignpar (-ignpar)	ignore (do not ignore) parity errors.
parmrk (-parmrk)	mark (do not mark) parity errors (see *termio*(7)).
inpck (—inpck)	enable (disable) input parity checking.
istrip (—istrip)	strip (do not strip) input characters to seven bits.
inlcr (—inlcr)	map (do not map) NL to CR on input.
igncr (—igncr)	ignore (do not ignore) CR on input.
icrnl (—icrnl)	map (do not map) CR to NL on input.
iuclc (—iuclc)	map (do not map) upper-case alphabetics to lower case on input.
ixon (—ixon)	enable (disable) START/STOP output control. Output is stopped by sending the stop character (default is CTRL-S) and started by sending the start character (default is CTRL-Q).
ixany (—ixany)	allow any character (only the start character like CTRL-Q) to restart output.
ixoff (—ixoff)	request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (—opost)	post-process output (do not post-process output; ignore all other output modes).
olcuc (—olcuc)	map (do not map) lower-case alphabetics to upper case on output.
onlcr (—onlcr)	map (do not map) NL to CR-NL on output.
ocrnl (—ocrnl)	map (do not map) CR to NL on output.
onocr (-onocr)	do not (do) output CRs at column zero.
onlret (-onlret)	on the terminal NL performs (does not perform) the CR function.
ofill (—ofill)	use fill characters (use timing) for delays.
ofdel (—ofdel)	fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	select style of delay for carriage returns (see *termio*(7)).
nl0 nl1	select style of delay for line-feeds (see *termio*(7)).
tab0 tab1 tab2 tab3	select style of delay for horizontal tabs (see *termio*(7)).
bs0 bs1	select style of delay for backspaces (see *termio*(7)).

ff0 ff1	select style of delay for form-feeds (see *termio*(7)).
vt0 vt1	select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (–isig)	enable (disable) the checking of characters against the special control characters INTR, QUIT and SWTCH.
icanon (–icanon)	enable (disable) canonical input (ERASE and KILL processing).
xcase (–xcase)	canonical (unprocessed) upper/lower-case presentation.
echo (–echo)	echo back (do not echo back) every character typed.
echoe (–echoe)	echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (–echok)	echo (do not echo) NL after KILL character.
lfkc (–lfkc)	the same as **echok (–echok)**; obsolete.
echonl (–echonl)	echo (do not echo) NL.
noflsh (–noflsh)	disable (enable) flush after INTR, QUIT or SWTCH.
stwrap (–stwrap)	disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (–stflush)	enable (disable) flush on a synchronous line after every *write*(2).
stappl (–stappl)	use application mode (use line mode) on a synchronous line.

Control Assignments

line *i*	set the line discipline to 0 (standard System V discipline) or 1 (4.3BSD *csh*(1) discipline).
control-character c	set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **ctab**, **min**, or **time** (**ctab** is used with **–stappl**; **min** and **time** are used with **–icanon**; see *termio*(7)). If line discipline is set to 1, the following *control-characters* can be set: **lnext**, **werase**, **rprnt**, **flush**, **stop**, **start**. If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^D" is a

CTRL-D); "^?" is interpreted as DEL and "^–" is interpreted as undefined.

Combination Modes

evenp or **parity**	enable **parenb** and **cs7**.
oddp	enable **parenb**, **cs7**, and **parodd**.
–parity, **–evenp**, or **–oddp**	disable **parenb**, and set **cs8**.
raw (**–raw** or **cooked**)	enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (**–nl**)	unset (set) **icrnl**, **onlcr**. In addition **–nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (**–lcase**)	set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (**–LCASE**)	same as **lcase** (**–lcase**).
tabs (**–tabs** or **tab3**)	preserve (expand to spaces) tabs when printing.
ek	reset ERASE and KILL characters back to normal ^H and ^U.
sane	resets all modes to some reasonable values.
term	set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1), ioctl(2), termio(7)
Extensions by Silicon Graphics, Inc.

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**–r**] file

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **–r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

sysadm – menu interface to do system administration

SYNOPSIS

sysadm [*sub-command*]

DESCRIPTION

This command, when invoked without an argument, presents a menu of system administration sub-commands, from which the user selects. If the optional argument is presented, the named sub-command is run or the named sub-menu is presented.

The *sysadm* command may be given a password. See **admpasswd** in the SUBCOMMANDS section.

SUB-COMMANDS

The following menus of sub-commands are available. (The number of bullets (•) in front of each item indicates the level of the menu or subcommand.)

• diagnostics

system diagnostics menu

These subcommands look for and sometimes repair problems in the system. Those subcommands that issue reports allow you to determine if there are detectable problems. Commands that attempt repair are for repair people only. You must know what you are doing!

•• diskrepair

advice on repair of built-in disk errors

This subcommand advises you on how to go about repairing errors that occur on built-in disks.

WARNING: Because this is a repair function, it should only be performed by qualified service personnel.
NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

•• diskreport

report on built-in disk errors

This subcommand shows you if the system has collected any information indicating that there have been errors while reading the built-in disks. You can request either summary or full reports. The summary report provides sufficient information about disk errors to determine if repair should be attempted. If the message no errors logged is part of the report, then there is probably no damage. If a number of errors is reported, there is damage and you should call for service. The full report gives additional detail for the expert repair person trouble shooting complicated problems. NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

•diskmgmt
disk management menu

The subcommands in this menu provide functions for using removable disks. The subcommands include the ability to format disks, copy disks, and to use disks as mountable file systems. It also contains a menu of subcommands for handling non-removable media.

•• checkfsys
check a removable disk file system for errors

Checkfsys checks a file system on a removable disk for errors. If there are errors, this procedure attempts to repair them.

•• cpdisk
make exact copies of a removable disk

This procedure copies the contents of a removable disk into the machine and then allows the user to make exact copies of it. These copies are identical to the original in every way. The copies are made by first reading the original removable disk entirely into the machine and then writing it out onto duplicate disks. The procedure will fail if there is not enough space in the system to hold the original disk.

•• erase
erase data from removable disk

This procedure erases a removable disk by overwriting it with null bytes. The main purpose is to remove data that the user does not want seen. Once performed, this operation is irreversible.

•• format
format new removable disks

Format prepares new removable disks for use. Once formatted, programs and data can be written on the disks.

●●harddisk

hard disk management menu

The subcommands in this menu provide functions for using hard disks. For each hard disk, the disk can be partitioned with default partitioning or the current disk partitioning can be displayed.

●●●display

display hard disk partitioning

Display will allow the user to display the hard disk partitioning. This will inform the user of current disk partitioning information.

●●●partitioning

partition a hard disk

Partitioning configures hard disks. This will allow you to partition a hard disk according to the default partitioning.

●●●rmdisk

remove a hard disk

Removes a hard disk from the system configuration. It may then be physically disconnected (once the machine has been turned off) or freshly partitioned (after the machine has been restarted).

●●makefsys

create a new file system on a removable disk

Makefsys creates a new file system on a removable disk which can then store data which the user does not wish to keep on the hard disk. When "mounted", the file system has all the properties of a file kept on the hard disk, except that it is smaller.

●●mountfsys

mount a removable disk file system

Mountfsys mounts a file system, found on a removable disk, making it available to the user. The file system is unmounted with the "umountfsys" command. THE DISK MUST NOT BE REMOVED WHILE THE FILE SYSTEM IS STILL MOUNTED.
IF THE FILE SYSTEM HAS BEEN MOUNTED WITH THE **mountfsys** COMMAND, IT MUST BE UNMOUNTED WITH **umountfsys**.

●• umountfsys
unmount a removable disk file system

Umountfsys unmounts a file system, allowing the user to remove the disk. THE DISK MUST NOT BE REMOVED UNTIL THE FILE SYSTEM IS UNMOUNTED.
umountfsys MAY ONLY BE USED TO UNMOUNT FILE SYSTEMS MOUNTED WITH THE **mountfsys** COMMAND.

● filemgmt
file management menu

The subcommands in this menu allow the user to protect files on the hard disk file systems by copying them onto diskettes and later restoring them to the hard disk by copying them back. Subcommands are also provided to determine which files might be best kept on diskette based on age or size.

●• diskuse
display how much of the hard disk is being used

Diskuse lets the user know what percentage of the hard disk is currently occupied by files. The list is organized by file system names.

●• fileage
list files older than a particular date

Fileage prints the names of all files older than the date specified by the user. If no date is entered, all files older than 90 days will be listed. If no directory is specified to look in, the **/usr/admin** directory will be used.

●• filesize
list the largest files in a particular directory

Filesize prints the names of the largest files in a specific directory. If no directory is specified, the **/usr/admin** directory will be used. If the user does not specify how many large files to list, 10 files will be listed.

● machinemgmt
machine management menu

Machine management functions are tools used to operate the machine, e.g., turn it off, reboot, or go to the firmware monitor.

●● autold
: set automatic boot device, default manual boot program

  This procedure specifies the default manual program to boot from firmware and/or the device to be used when automatically rebooting.

●● firmware
: stop all running programs then enter firmware mode

  This procedure will stop all running programs, close any open files, write out information to the disk (such as directory information), then enter the firmware mode. (Machine diagnostics and other special functions that are not available on the UNIX system.)

●● floppykey
: create a "floppy key" removable disk

  The "floppy key" removable disk allows the user to enter firmware mode if the firmware password has been changed and then forgotten. Thus the "floppy key" is just that, the "key" to the system and should be protected as such.

●● powerdown
: stop all running programs, then turn off the machine

  Powerdown will stop all running programs, close any open files, write out information to disk (such as directory information), then turn the machine power off.

●● reboot
: stop all running programs then reboot the machine

  Reboot will stop all running programs, close any open files, write out information to disk (such as directory information), then reboot the machine. This can be used to get out of some types of system trouble, such as when a process cannot be killed.

●● whoson
: print list of users currently logged onto the system

  Whoson prints the login ID, terminal device number, and sign-on time of all users who are currently using the computer.

● packagemgmt
  package management

  These submenus and subcommands manage various software and hardware packages that you install on your machine. Not all optional packages add subcommands here.

● softwaremgmt
  software management menu

  These subcommands permit the user to install new software, remove software, and run software directly from the removable disk it is delivered on. The "remove" and "run" capabilities are dependent on the particular software packages. See the instructions delivered with each package.

●•installpkg
  install new software package onto integral hard disk

  Install copies files from removable disk onto the integral hard disk and performs additional work if necessary so that the software can be run. From then on, the user will have access to those commands.

●•listpkg
  list packages already installed

  This subcommand show you a list of currently installed optional software packages.

●•removepkg
  remove previously installed package from integral hard disk

  This subcommand displays a list of currently installed optional software packages. Actions necessary to remove the software packages specified by the user will then be performed. The removable disk used to "installpkg" the software is needed to remove it.

●•runpkg
  run software package without installing it

  This package allows the user to run software from a removable disk without installing it permanently on the system. This is useful if the user does not use the software often or does not have enough room on the system. WARNING: Not all software packages have the ability to run their contents this way. See the instructions that come with the software package.

• syssetup

system setup menu

System setup routines allow the user to tell the computer what its environment looks like: what the date, time, and time zone is, what administration and system capabilities are to be under password control, what the machine's name is, etc. The first-time setup sequence is also here.

•• admpasswd

assign or change administrative passwords

Admpasswd lets you set or make changes to passwords for administrative commands and logins such as setup and sysadm.

•• datetime

set the date, time, time zone, and daylight savings time

Datetime tells the computer the date, time, time zone, and whether you observe Daylight Savings Time (DST). It is normally run once when the machine is first set up. If you observe DST, the computer will automatically start to observe it in the spring and return to Standard Time in the fall. The machine has to be turned off and turned back on again to guarantee that ALL times will be reported correctly. Most are correct the next time the user logs in.

•• nodename

set the node name of this machine

This allows you to change the node name of this machine. The node name is used by various communications networks to identify this machine.

•• setup

set up your machine the very first time

Setup allows the user to define the first login, to set the passwords on the user-definable administration logins and to set the time zone for your location.

•• syspasswd

assign system passwords

Syspasswd lets the user set system passwords normally reserved for the very knowledgeable user. For this reason, this procedure may assign those passwords, but may not change or clear them. Once set, they may only be changed by the specific login or the "root" login.

● ttymgmt
terminal management

This procedure allows the user to manage the computer's terminal functions.

●● lineset
show tty line settings and hunt sequences

The tty line settings are often hunt sequences where, if the first line setting does not work, the line "hunts" to the next line setting until one that does work comes by. This subcommand shows the various sequences with only specific line settings in them. It also shows each line setting in detail.

●● mklineset
create new tyy line settings and hunt sequences

This subcommand helps you to create tty line setting entries. You might want to add line settings that are not in the current set or create hunt sequences with only specific line settings in them. The created hunt sequences are circular; stepping past the last setting puts you on the first.

●● modtty
show and optionally modify characteristics of tty lines

This subcommand reports and allows you to change the characteristics of tty lines (also called "ports").

● usermgmt
user management menu

These subcommands allow you to add, modify and delete the list of users that have access to your machine. You can also place them in separate groups so that they can share access to files within the group but protect themselves from other groups.

●● addgroup
add a group to the system

Addgroup adds a new group name or ID to the computer. Group names and IDs are used to identify groups of users who desire common access to a set of files and directories.

●● adduser
add a user to the system

Adduser installs a new login ID on the machine. You are asked a series of questions about the user and then the new entry is made. You can enter more than one user at a time. Once this procedure is finished, the new login ID is available.

●● delgroup
delete a group from the system

Delgroup allows you to remove groups from the computer. The deleted group is no longer identified by name. However, files may still be identified with the group ID number.

●● deluser
delete a user from the system

Deluser allows you to remove users from the computer. The deleted user's files are removed from the hard disk and their logins are removed from the **/etc/passwd** file.

●● lsgroup
list groups in the system

Lsgroup will list all the groups that have been entered into the computer. This list is updated automatically by "addgroup" and "delgroup"

●● lsuser
list users in the system

Lsuser will list all the users that have been entered into the computer. This list is updated automatically by "adduser" and "deluser".

●● modadduser
modify defaults used by adduser

Modadduser allows the user to change some of the defaults used when adduser creates a new login. Changing the defaults does not effect any existing logins, only logins made from this point on.

●● modgroup

make changes to a group on the system

Modgroup allows the user to change the name of a group that the user enters when "addgroup" is run to set up new groups.

●● moduser

menu of commands to modify a user's login

This menu contains commands that modify the various aspects of a user's login.

●●● chgloginid

change a user's login ID

This procedure allows the user to change a user's login ID. Administrative and system logins cannot be changed.

●●● chgpasswd

change a user's passwd

This proceudure allows removal or change of a suer's password. Administrative and system login passwords channot be changed. To change administrative and system login passwords, see the system setup menu: sysadm syssetup.

●●● chgshell

change a user's login shell

This procedure allows the user to change the command run when a user logs in. The login shell of the administrative and system logins cannot be changed by this procedure.

EXAMPLES

sysadm adduser

FILES

The files that support *sysadm* are found in **/usr/admin.**

The menu starts in directory **/usr/admin/menu.**

NAME

sysinfo – print system identification

SYNOPSIS

sysinfo [-s]

DESCRIPTION

sysinfo with no options prints the unique identifier of the system. This identifier is guaranteed to be unique within the Silicon Graphics product family. With the -s option a shorter (32 bit) identifier is printed. This identifier is not guaranteed to be unique but in practice is. This number is the same that is available from within a program via the *sysid*(3C) function.

SEE ALSO

sysid(3C).

BUGS

The system identifier on different Silicon Graphics products is associated with different physical pieces of hardware. Therefore, the identifier may unexpectedly change when the hardware is modified.

NAME

sysmeter – meter display of system performance values

SYNOPSIS

sysmeter [**–s** *smp*] [**–a** *avg*] [**–h** *name*] [**–d** *wdw*] [**–g** *ht*] [**–x** *x-cord*] [**–y** *y-cord*] [**–r** *row*] [**–m**] [**–i** *color*] [**–v** *value value* ...]

DESCRIPTION

Sysmeter opens a window and displays various system performance values collected from either local machine, or any remote machine that implements the *rstatd(1M)*, version 3. When *sysmeter* starts, it makes a connection to the *rstatd(1M)* daemon on the target machine and have this daemon report back the performance data collected from its local operating system periodically.

Various performance data can be selected for display by user's options. Each selected performance value will be painted within one sub-window in either of the two styles: bar chart or strip chart. In both charts, the y-axis represents the value collected. In strip chart, the x-axis represents the time.

The default option is to collect cpu usage data on local machine for every 2 sec. with window width equals to 120 sec. The options can be set either from command line or by clicking the left or right mouse button.

User also can choose whether to display the average of performance values. The averaging values are displayed as shaded area in the strip chart, and it appears as the rear bar in bar chart.

When resizing the window, *sysmeter* always re-adjusts the window size for the best possible shape to fit all the selected sub-windows in order to prevent the distortion of the graphics. If the width of the window is wide enough, *sysmeter* will display hostname, sample time, averaging time and window width on top of the window,

OPTIONS

–s *smp*	Sets the sample time to *smp* seconds.
–a *avg*	Sets the averaging time to *avg* seconds. If this option is not supplied, *sysmeter* will not display the averaging value.
–h *name*	Selects host *name* as the target machine for performance data collection.

−d *wdw*	Sets window width to *wdw* seconds. This number divided by the sample time is equivalent to the number of samples appears in one sub-window. This option is meaningful only in strip chart.
−g *ht*	Sets the sub-window height to *ht* pixels. The default height is 50 pixels.
−x *x-cord*	Sets the x coordinate of the left lower corner of the window to *x-cord*. Since *sysmeter* always re-adjust window size and location on a bad re-locating or resizing, a large x coordinate close to the left end of screen will automatically be reduced as needed.
−y *y-cord*	Sets the y coordinate of the left lower corner of the window to *y-cord*. If both x and y coordinates are provided, *sysmeter* will automatically open a window without user intervention.
−r *row*	Sets the number of rows of sub-windows to be *row*. The default is two columns.
−m	displays bar chart instead of the default strip chart.
−i *color*	The *color* Sets the first of the five contiguous color indexs used for graphical display. The default value is 0;
−v *value*	Selects performance values to be displayed. Valid values are:

cpu	percent of cpu being utilized
pkts	ethernet packets per second
page	paging activity in pages per second
swap	jobs swapped per second
intr	number of device interrupts per second
disk	disk traffic in transfers per second
cntxt	number of context switches per second
load	average number of jobs running on the server over the last minute

colls	collisions per second detected on the ethernet
errs	errors per second on receiving packets

When *sysmeter* is running, user can move the mouse cursor into the window, press and hold the right button to get the pop-up menu. This menu has the options: *cpu, pkts, page, swap, intr, disk, display-average, change-param, style, and exit.* The entries corresponding to each performance values specified in -v option are toggle switches. If user moves the cursor to any one of those entries and releases the right button, *sysmeter* will toggle its internal selection switch and the sub-window will be either displayed or removed. if no parameters are specified when *sysmeter* is started, it will display the: *cpu* usage by default. The *display-average* and *style* selections are also toggle switches.

When *change-parameter* entry is selected, a parameter changing panel will appear in the left lower corner of the window and the display activity is frozen although *sysmeter* is still collecting data packets internally. If the window is too narrow, only partial of the panel will show up. User can fix this problem by making more value selection or reshape the window.

Inside the panel, there is a parameter selection location marked with three curly arrows. Moving the cursor to this area and click the left button, the panel will switch among the three changeable parameters: *sampl* - sample interval, *averg* - averaging period, and *wndow* - window width.

After moving the cursor to the up-pointed triangle under the ruler, pressing, holding and moving the left button along the ruler will change the selected parameter to the number indicated in the small window above the ruler. The ruler is exponentially increased to maximum of 7200 seconds (2 hours).

Once the change is made, moving the cursor to the location marked with *ok* and press the left button, the parameter will be picked up by *sysmeter* and the window is re-adjusted as necessary.

EXAMPLE

The following command line will automatically displays the local system performance data on the top of the screen.

sysmeter -x 165 -y 930 -v cpu pkts page swap intr disk cntxt load colls errs -a 120 -g 50 -d 300 -s 5 -r 1

SEE ALSO

rstatd(1M), rpcinfo(1)

BUG

The maximum number allowed to be displayed on the y-axis of a subwindow is 8192. If value exceeds this limit, the display will be truncated to 8192.

AUTHOR

Steve Sun

NAME

systemdown – interactive script for shutting the system down.

SYNOPSIS

systemdown

DESCRIPTION

The *systemdown* command is an interactive command used for shutting the system down. Depending on who invokes *systemdown,* a prompt will appear to either request for the root password or for confirmation of the system shutdown. Upon confirmation, *systemdown* will then execute *init 0.*

SEE ALSO

init(1)

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [tabspec] [–Ttype] [+mn]

DESCRIPTION

tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*–code*), repetitive (*–n*), arbitrary (*n1,n2,...*), and file (*––file*). If no *tabspec* is given, the default value is **–8**, i.e., UNIX system "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

–code Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

–a	1,10,16,36,72 Assembler, IBM S/370, first format
–a2	1,10,16,40,72 Assembler, IBM S/370, second format
–c	1,8,12,16,20,55 COBOL, normal format
–c2	1,6,10,14,49 COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see *fspec*(4)): **<:t–c2 m6 s66 d:>**
–c3	1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67 COBOL compact format (columns 1-6 omitted), with more tabs than **–c2**. This is the recommended format for COBOL. The appropriate format specification is (see *fspec*(4)): **<:t–c3 m6 s66 d:>**
–f	1,7,11,15,19,23 FORTRAN
–p	1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61 PL/I

–s	1,10,55
	SNOBOL
–u	1,12,20,44
	UNIVAC 1100 Assembler

–n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **8**: this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.

n1,*n2*,... The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30**, and **1,10,+10,+10** are considered identical.

—file If the name of a *file* is given, *tabs* reads the first line of the file, searching for a format specification (see *fspec*(4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as **–8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

```
tabs — file; pr file
```

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

–T*type* *tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(5). If no **–T** flag is supplied, *tabs* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* (see *environ*(5)), *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10**. For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

EXAMPLES

tabs –a example using *–code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs –8 example of using *–n* (*repetitive* specification), where *n* is 8, causes tabs to be set every eighth position:
1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36 example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs —$HOME/fspec.list/att4425
example of using *—file* (*file* specification) to indicate that tabs should be set according to the first line of *$HOME/fspec.list/att4425* (see *fspec*(4)).

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly
illegal increment	when a zero or missing increment is found in an arbitrary specification
unknown tab code	when a *canned* code cannot be found
can't open	if *—file* option used, and file can't be opened
file indirection	if *—file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

NOTES

Hardware tabs must be enabled on the terminal device by entering the UNIX command *'stty tabs'*; otherwise the *tabs* command will appear to have no effect. Tab and margin setting is performed via the standard output.

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The *tabspec* used with the *tabs* command is different from the one used with the *newform*(1) command. For example, **tabs –8** sets every eighth position; whereas **newform –i–8** indicates that tabs are set every eighth position.

SEE ALSO

stty(1), newform(1), pr(1), tput(1).
fspec(4), terminfo(4), environ(5), term(5) in the *Programmer's Reference Manual*.

NAME

tag – tag a MIPS executable or shell script with an identifying number

SYNOPSIS

tag number filename[s]
tag –c filename[s]
tag filename
tag -q filenames

DESCRIPTION

tag is used to set, clear or query the tag number in a MIPS executable, or shell script that follows the convention of #!/bin/sh or #!/bin/csh on the first line. The tag number is used by the SGI WorkSpace to determine the type of a file.

tag number filename[s]	sets the tag number of a MIPS executable or script. (Many executables or scripts can be specified to be tagged with the same number.) The number must be non-negative and less than 4294967296.
tag –c filename[s]	clears the tags on the specified file or files.
tag filename	prints out the tag number of a MIPS executable or script.
tag –q filenames	prints out the tag numbers of a list of MIPS executables or scripts.

Tag numbers are administered by the *Silicon Graphics* user interface group. Contact them in order to get a block of tag numbers.

The tag number is stored as a longword in bytes 68 through 71 (numbering from zero) of the MIPS executable. The most significant bit of byte 18 is set when the file has been tag and will not be set otherwise. For shell scripts, the line '#Tag <number>' will be inserted as the second line of the shell script.

BUGS

Only shell scripts whose first line is exactly "#!/bin/sh" or "!#/bin/csh", with no trailing flags on the first line are recognized by the tag command.

SEE ALSO

Programming the IRIS WorkSpace

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *–number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **–f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail –f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail –15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1M).

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The *tail* command will only tail the last 256 Kbytes of a file regardless of its line count.

NAME

talk – talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

Talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form *user@host*.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

```
talk your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control L will cause the screen to be reprinted, while your erase, kill, and word kill characters will work in talk as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1)

BUGS

This version of *talk*(1) is incompatible with the protocol used in the version released with 4.2BSD.

NAME

tar – tape archiver

SYNOPSIS

tar key [name ...]

DESCRIPTION

tar saves and restores multiple files on a single file (usually a magnetic tape, but it can be any file). The *key* argument controls *tar*'s actions. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to *tar* are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers (recursively) to the files and subdirectories of that directory. These files are dumped to tape in alphabetical order.

A *tar* archive is a stream of 512-byte header structures which may be followed by file data rounded up to the next 512-byte boundary. The end of the archive is signaled by two header structures beginning with null bytes.

The function portion of the key is specified by one of the following letters:

r	Append the named files at the end of the archive. On tape, named files are appended at the end of the last archive on tape. This function is only supported on half-inch nine track tape drives. It is not supported on any variety of quarter inch cartridge tape drive or on the Exabyte 8mm tape drive, because the drive hardware does not allow this function to be implemented.
x	Extract the named files from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last entry overwrites all earlier entries.
X	For each file to be extracted, if it is identical to the file in the corresponding position in the comparison tree, link the existing file to the new file. Otherwise, extract the new file as a separate new file. X is like x but also takes the next argument as the root of a directory tree for comparison.
t	List the names of the specified files each time they occur on the tape. If no file argument is given, list all of the names on the tape.
c	Create a new tape; writing starts at the beginning of the tape instead of after the last file. This option assumes that you are at the beginning of the tape.

C Compare files on tape against existing files. For each specified file, print a line with a key character followed by the file name.

L	linked to an earlier file on the tape
S	symbolic link
B	block special file
C	character special file
P	named pipe
?	can't read the disk file, so can't compare
>	disk file doesn't exist
=	files compare
!	files don't compare

In addition to the letter which selects the function desired the following characters may be used:

d On output, *tar* normally places information specifying owner and modes of directories in the archive. Former versions of *tar*, when encountering this information print error messages of the form:

''<name>/: cannot create''.

This option suppresses the printing of the directory information. This option implies the **D** option.

D On output, *tar* normally places information specifying the owner, modes, and device numbers of character and block special files and named pipes (*fifos*) in the archive. Former versions of *tar*, when encountering this information create an ordinary file of the same name whose contents is the device number, in binary. This option suppresses the special file information.

p This option restores files to their original modes, ignoring the present *umask*(2). Setuid and sticky information are also restored to the super-user.

v Normally *tar* does its work silently. The v (verbose) option make *tar* print the name of each file it treats preceded by the function letter. With the **t** function, the verbose option gives more information about the tape entries than just their names.

w *tar* prints the action to be taken followed by file name, then waits for user confirmation. If a word beginning with 'y' is given, the action is done. Any other input means do not do it.

f *tar* uses the next argument as the name of the archive instead of */dev/tape*. If the name of the file is '–', tar writes to standard output or reads from standard input (whichever is appropriate). Thus, *tar* can be used as the head or tail of a filter chain.

The specified archive name can also reference a remote tape device. A remote tape device name has the form:

*[user@]*system:/dev/???

where *system* is the remote system, */dev/???* is the particular drive on the remote system (raw, rewinding, non-rewinding, etc.), and the optional *user* is the login name to be used on the remote system, if different from the current user's login name.

Tar can also be used to move hierarchies on the local machine with the command

```
cd fromdir; tar cBf - . | (cd todir; tar xBf -)
```

To move hierarchies between machines, use the command

```
cd fromdir; tar cBf - . | rsh remote "(cd todir; tar xBf -)"
```

where *remote* is the host name of the remote machine.

b *tar* uses the next argument as the blocking factor for tape records. The default is 400 for the cartridge tape, 1 for standard input and standard output, and 20 otherwise. This option should only be used with raw magnetic tape archives (See **f** above) or the default tape device. If the tape was written with a blocking factor that does not exceed the default for that device (20 or 400), the block size is determined automatically when reading a tape. Use the default blocking factor with a cartridge tape. Due to the blocking algorithm, a tar tape created by writing to the standard output should be read from standard input.

l If it cannot resolve all of the links to the files dumped, *tar* prints error messages. If **l** is not specified, no error messages are printed.

m Do not restore the modification times. The modification time will be the time of extraction.

e Force *tar* to continue reading past tape errors.

L Force *tar* to follow symbolic links as if they were normal files or directories.

B Force input and output blocking to 20 blocks per record. This option allows *tar* to work across a communications channel where the blocking may not be maintained.

R When extracting from tape, ignore leading slashes on file names, i.e., extract all files relative to the current directory.

U For each file extracted, unlink existing file (if any).

o Do not *chown* (or *chgrp*) files.

a Reset access times of input files after they have been copied to the archive.

q Turn on debugging and extra error diagnostics. Supplying this flag multiple times increases debugging level.

If a file name is preceded by **−C**, then *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from */usr/include* and from */etc*, one might use

```
tar c −C /usr  include −C /  etc
```

If a file name of − is given on the command line when making an archive then *tar* will read its standard input for a list of files to back up, one per line; the list is terminated by an EOF. For example, to back up all files that have changed in the last week, one might use

```
find /  −local  !  −type  d  −mtime  −7 −type  f −print | tar  ca  −
```

FILES

/dev/tape
/dev/mt/*
/tmp/tar*

SEE ALSO

mtio(7)

DIAGNOSTICS

Error messages about bad key characters and tape read/write errors.

Error messages if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*−th occurrence of a file.

The **u** option can be slow.

File name length is limited to 100 characters.

The data for a file with multiple links is output to tape with the first link encountered. Thus, an attempt to extract a subsequent link by itself will not have the desired result.

The **r** command works only on half inch nine track tape drives. It is not supported on any variety of quarter inch cartridge tape or on the Exabyte 8mm tape drives, because the controllers and drives don't allow this function to be implemented. For the **r** command to work on a multi-partition tape, one has to position the tape head to the last partition on the tape. See *mt*(1) for more information about how to append a new archive at the end of recorded data.

If you open a remote tape device and receive an I/O error, it is probable that the remote tape interface program is obsolete and not compatible with the remote tape subroutine library that tar uses.

NAME

tarArchive – an interactive *transferdevice* for performing tar within the WorkSpace.

SYNOPSIS

tarArchive menu
tarArchive versionOK
tarArchive import
tarArchive export
tarArchive dir

DESCRIPTION

The *tarArchive* command is an interactive tool used for copying files to or from the WorkSpace and a tape drive using tar. It is a type of *transferdevice*, and therefore understands the generic arguments **menu** and **versionOK**.

The argument **import** is used to copy files from a *tar* format tape to the WorkSpace's currently selected directory. **export** is used to copy the files currently selected in the WorkSpace to a tape. When given the argument **dir**, *tarArchive* will produce a listing of the files on the tape. The *tarArchive* examines the environment variable $SELECTED, which is set to be the current selection by the WorkSpace. *tarArchive* can be used to read or write tapes on a remote machine.

FILES

/etc/transferDevice/

SEE ALSO

transfermanager(1G), transferdevice(4), rcpDevice(1), cpioArchive(1), workspace(1G), tar(1)
Programming the IRIS WorkSpace

NAME

tee – pipe fitting

SYNOPSIS

tee [**–i**] [**–a**] [file] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files* . The

–i	ignore interrupts;
–a	causes the output to be appended to the *files* rather than overwriting them.

NAME

telnet – User interface to the TELNET protocol

SYNOPSIS

telnet [**–d**] [**–n** *tracefile*] [*host* [*port*]]

DESCRIPTION

The **telnet** command is used to communicate with another host using the TELNET protocol. If **telnet** is invoked without the *host* argument, it enters command mode, indicated by its prompt (**telnet>**). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, *telnet* will attempt to enable the TELNET LINEMODE option. If this fails, then *telnet* will revert to one of two input modes: either ''character at a time'' or ''old line by line'' depending on what the remote system supports.

When LINEMODE is enabled, character processing is done on the local system, under the control of the remote system. When input editing or character echoing is to be disabled, the remote system will relay that information. The remote system will also relay changes to any special characters that happen on the remote system, so that they can take effect on the local system.

In ''character at a time'' mode, most text typed is immediately sent to the remote host for processing.

In ''old line by line'' mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The ''local echo character'' (initially ''^E'') may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

If the LINEMODE option is enabled, or if the **localchars** toggle is TRUE (the default for ''old line by line''; see below), the user's **quit**, **intr**, and **flush** characters are trapped locally, and sent as TELNET protocol sequences to the remote side. If LINEMODE has ever been enabled, then the user's **susp** and **eof** are also sent as TELNET protocol sequences, and **quit** is sent as a TELNET ABORT instead of BREAK. There are options (see **toggle autoflush** and **toggle autosynch** below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, *telnet* command mode may be entered by typing the *telnet* ''escape character'' (initially ''^]''). When in command mode, the normal terminal editing conventions are available.

Options:

−d Sets the initial value of the **debug** toggle to TRUE.

−n *tracefile*
: Opens *tracefile* for recording trace information. See the set **tracefile** command below.

host Indicates the official name, an alias, or the Internet address of a remote host.

port Indicates a port number (address of an application). If a number is not specified, the default *telnet* port is used.

The following *telnet* commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, **unset**, **slc**, and **display** commands).

close
: Close a TELNET session and return to command mode.

display [*argument...*]
: Displays all, or some, of the **set** and **toggle** values (see below).

mode *type*
: *Type* is one of several options, depending on the state of the TELNET session. The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

  character
  : Disable the TELNET LINEMODE option, or, if the remote side does not understand the LINEMODE option, then enter ''character at a time'' mode.

  line
  : Enable the TELNET LINEMODE option, or, if the remote side does not understand the LINEMODE option, then attempt to enter ''old-line-by-line'' mode.

  isig (−isig)
  : Attempt to enable (disable) the TRAPSIG mode of the LINEMODE option. This requires that the LINEMODE option be enabled.

  edit (−edit)
  : Attempt to enable (disable) the EDIT mode of the LINEMODE option. This requires that the LINEMODE option be enabled.

?

Prints out help information for the **mode** command.

open *host* [[−]*port*]

Open a connection to the named host. If no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see **hosts**(4)) or an Internet address specified in the ''dot notation'' (see **inet**(3N)). When connecting to a non-standard port, *telnet* omits any automatic initiation of TELNET options. When the port number is preceded by a minus sign, the initial option negotiation is done. After establishing a connection, the **.telnetrc** in the users home directory is opened. Lines beginning with a # are comment lines. Blank lines are ignored. Lines that begin without whitespace are the start of a machine entry. The first thing on the line is the name of the machine that is being connected to. The rest of the line, and successive lines that begin with whitespace are assumed to be **telnet** commands and are processed as if they had been typed in manually to the **telnet** command prompt.

quit

Close any open TELNET session and exit **telnet**. An end of file (in command mode) will also close a session and exit.

send *arguments*

Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

abort

Sends the TELNET ABORT (ABORT processes) sequence.

ao

Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output from the remote system to the user's terminal.

ayt

Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

brk

Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

ec
: Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el
: Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

eof
: Sends the TELNET EOF (End Of File) sequence.

eor
: Sends the TELNET EOR (End of Record) sequence.

escape
: Sends the current **telnet** escape character (initially "^]").

ga
: Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

getstatus
: If the remote side supports the TELNET STATUS command, **getstatus** will send the subnegotiation to request that the server send its current option status.

ip
: Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

nop
: Sends the TELNET NOP (No OPeration) sequence.

susp
: Sends the TELNET SUSP (SUSPend process) sequence.

synch
: Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system — if it doesn't work, a lower case "r" may be echoed on the terminal).

?

Prints out help information for the **send** command.

set *argument value*

unset *arguments...*

The **set** command will set any one of a number of **telnet** variables to a specific value or to TRUE. The special value **off** turns off the function associated with the variable, this is equivalent to using the **unset** command. The **unset** command will disable or set to FALSE any of the specified functions. The values of variables may be interrogated with the **display** command. The variables which may be set or unset, but not toggled, are listed here. In addition, any of the variables for the **toggle** command may be explicitly set or unset using the **set** and **unset** commands.

echo

This is the value (initially "^E") which, when in "line by line" mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

eof

If **telnet** is operating in LINEMODE or "old line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

erase

If **telnet** is in *localchars* mode (see **toggle localchars** below), **and** if **telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET EC sequence (see **send ec** above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

escape

This is the **telnet** escape character (initially "^[") which causes entry into **telnet** command mode (when connected to a remote system).

flushoutput

If **telnet** is in *localchars* mode (see **toggle localchars** below) and the **flushoutput** character is typed, a TELNET AO sequence (see **send ao** above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

interrupt
If **telnet** is in *localchars* mode (see **toggle localchars** below) and the **interrupt** character is typed, a TELNET IP sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

kill
If **telnet** is in *localchars* mode (see **toggle localchars** below), **and if telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET EL sequence (see **send el** above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

lnext If **telnet** is operating in LINEMODE or "old line by line" mode, then this character is taken to be the terminal's **lnext** character. The initial value for the lnext character is taken to be the terminal's **lnext** character.

quit
If **telnet** is in *localchars* mode (see **toggle localchars** below) and the **quit** character is typed, a TELNET BRK sequence (see **send brk** above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

reprint
If **telnet** is operating in LINEMODE or "old line by line" mode, then this character is taken to be the terminal's **reprint** character. The initial value for the reprint character is taken to be the terminal's **reprint** character.

start
If the TELNET TOGGLE-FLOW-CONTROL option has been enabled, then this character is taken to be the terminal's **start** character. The initial value for the kill character is taken to be the terminal's **start** character.

stop

If the TELNET TOGGLE-FLOW-CONTROL option has been enabled, then this character is taken to be the terminal's **stop** character. The initial value for the kill character is taken to be the terminal's **stop** character.

susp

If **telnet** is in **localchars** mode, or LINEMODE is enabled, and the **suspend** character is typed, a TELNET SUSP sequence (see **send susp** above) is sent to the remote host. The initial value for the suspend character is taken to be the terminal's **suspend** character.

tracefile

This is the file to which the output, caused by **netdata** or **option** tracing being TRUE, will be written. If it is set to '–', then tracing information will be written to standard output (the default).

worderase

If **telnet** is operating in LINEMODE or ''old line by line'' mode, then this character is taken to be the terminal's *worderase* character. The initial value for the worderase character is taken to be the terminal's *worderase* character.

slc *state*

The slc command (Set Local Characters) is used to set or change the state of the the special characters when the TELNET LINEMODE option has been enabled. Special characters are characters that get mapped to TELNET commands sequences (like **ip** or **quit**) or line editing characters (like **erase** and **kill**). By default, the local special characters are exported.

export

Switch to the local defaults for the special characters. The local default characters are those of the local terminal at the time when **telnet** was started.

import

Switch to the remote defaults for the special characters. The remote default characters are those of the remote system at the time when the TELNET connection was established.

check

Verify the current settings for the current special characters. The remote side is requested to send all the current special character settings, and if there are any discrepancies with the local side, the local side will switch to the remote value.

?

Prints out help information for the **slc** command.

?

Displays the legal **set** (**unset**) commands.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how **telnet** responds to events. These flags may be set explicitly to TRUE or FALSE using the **set** and **unset** commands listed above. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

autoflush

If **autoflush** and **localchars** are both TRUE, then when the ao, **intr**, or **quit** characters are recognized (and transformed into TELNET sequences; see **set** above for details), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET TIMING MARK option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE (see stty(1)).

autosynch

If **autosynch** and **localchars** are both TRUE, then when either the **intr** or **quit** characters is typed (see **set** above for descriptions of the **intr** and **quit** characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

binary
Enable or disable the TELNET BINARY option on both input and output.

inbinary
Enable or disable the TELNET BINARY option on input.

outbinary
Enable or disable the TELNET BINARY option on output.

crlf
If this is TRUE, then carriage returns will be sent as <CR><LF>. If this is FALSE, then carriage returns will be send as <CR><NUL>. The initial value for this toggle is FALSE.

crmod
Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

debug
Toggles socket level debugging (useful only to the *super*user). The initial value for this toggle is FALSE.

localchars
If this is TRUE, then the **flush**, **interrupt**, **quit**, **erase**, and **kill** characters (see set above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively **ao**, **ip**, **brk**, **ec**, and **el**; see **send** above). The initial value for this toggle is TRUE in "old line by line" mode, and FALSE in "character at a time" mode. When the LINEMODE option is enabled, the value of **localchars** is ignored, and assumed to always be TRUE. If LINEMODE has ever been enabled, then **quit** is sent as **abort**, and **eof**and **suspend** are sent as **eof**and **susp**, see **send** above).

netdata

Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

options

Toggles the display of some internal **telnet** protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

prettydump

When the **netdata** toggle is enabled, if **prettydump** is enabled the output from the **netdata** command will be formatted in a more user readable format. Spaces are put between each character in the output, and the beginning of any TELNET escape sequence is preceded by a '*' to aid in locating them.

?

Displays the legal **toggle** commands.

z

Suspend **telnet**. This command only works when the user is using the **csh**(1).

! [*command*]

Execute a single command in a subshell on the local system. If *command* is omitted, then an interactive subshell is invoked.

status

Show the current status of **telnet**. This includes the peer one is connected to, as well as the current mode.

? [*command*]

Get help. With no arguments, **telnet** prints a help summary. If a command is specified, **telnet** will print the help information for just that command.

FILES

~/.telnetrc

NOTES

On some remote systems, echo has to be turned off manually when in "old line by line" mode.

In "old line by line" mode or LINEMODE the terminal's *eof* character is only recognized (and sent to the remote system) when it is the first character on a line.

NAME

test – condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; *test* also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS line) must be separate arguments to the *test* command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

–r *file*	true if *file* exists and is readable.
–l *file*	true if *file* exists and is a symbolic link.
–w *file*	true if *file* exists and is writable.
–x *file*	true if *file* exists and is executable.
–f *file*	true if *file* exists and is a regular file.
–d *file*	true if *file* exists and is a directory.
–c *file*	true if *file* exists and is a character special file.
–b *file*	true if *file* exists and is a block special file.
–p *file*	true if *file* exists and is a named pipe (fifo).
–u *file*	true if *file* exists and its set-user-ID bit is set.
–g *file*	true if *file* exists and its set-group-ID bit is set.
–k *file*	true if *file* exists and its sticky bit is set.
–s *file*	true if *file* exists and has a size greater than zero.
–t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–z *s1*	true if the length of string *s1* is zero.
–n *s1*	true if the length of the string *s1* is non-zero.
s1 = *s2*	true if strings *s1* and *s2* are identical.

s1 != *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **−eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of −eq.

These primaries may be combined with the following operators:

!	unary negation operator.
−a	binary *and* operator.
−o	binary *or* operator (−a has higher precedence than **−o**).
(expr)	parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the *-r*, *-w*, or *-x* tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The = and != operators have a higher precedence than the **−r** through **−n** operators, and = and != always expect arguments; therefore, = and != cannot be used with the **−r** through **−n** operators.

If more than one argument follows the **−r** through **−n** operators, only the first argument is examined; the others are ignored, unless a **−a** or a **−o** is the second argument.

NAME

textcolors – set the colors used by a text window

SYNOPSIS

textcolors text page reverse cursor [selfg selbg]

DESCRIPTION

There are several color indices associated with each text window. *textcolors* sets the color index to be used for printed text, the page color, reverse video text, and the cursor. Optionally, the selection foreground and selection background colors can be set. If the selection colors are not specified, the old values are used. The color indices given must be between 0 and 255.

EXAMPLES

textcolors 0 7 4 4

Let's assume that color index 0 is black, 7 is white and 4 is blue. This command will make text print black on a white background. Highlighted text and the cursor will be displayed in blue.

textcolors 0 7 4 4 7 0

With the same assumptions as above, this makes the selection show as white text on a black background.

NAME

tftp – trivial file transfer program

SYNOPSIS

tftp [host]

DESCRIPTION

Tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case *tftp* uses *host* as the default host for future transfers (see the **connect** command below).

COMMANDS

Once *tftp* is running, it issues the prompt **tftp>** and recognizes the following commands:

connect *host-name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the *connect* command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the *connect* command; the remote host can be specified as part of the *get* or *put* commands.

mode *transfer-mode*
: Set the mode for transfers; *transfer-mode* may be one of *ascii* or *binary*. The default is *ascii*.

put *file*

put *localfile remotefile*
put *file1 file2 ... fileN remote-directory*
: Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a *UNIX* machine.

get *filename*
get *remotename localname*
get *file1 file2 ... fileN*
: Get a file or set of files from the specified *sources*. *Source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter

form is used, the last hostname specified becomes the default for future transfers.

quit Exit *tftp*. An end of file also exits.

verbose Toggle verbose mode.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*
Set the total transmission timeout, in seconds.

ascii Shorthand for "mode ascii"

binary Shorthand for "mode binary"

? [*command-name* ...]
Print help information.

BUGS

Because there is no user-login or validation within the *TFTP* protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

times(2) in the *Programmer's Reference Manual.*

NAME

timex – time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

–p List process accounting records for *command* and all its children. Suboptions **f, h, k, m, r,** and **t** modify the data items reported. The options are as follows:

> **–f** Print the *fork*/*exec* flag and system exit status columns in the output.
>
> **–h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
>
> (total CPU time)/(elapsed time).
>
> **–k** Instead of memory size, show total kcore-minutes.
>
> **–m** Show mean core size (the default).
>
> **–r** Show CPU factor (user time/(system-time + user-time).
>
> **–t** Show separate system and user CPU times. The number of blocks read or written and the number of characters transferred are always reported.

–o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

–s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
                timex –ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
        timex –opskmt sh

                session commands
        EOT
```

NAME

tlink – clone a file tree using symbolic links

SYNOPSIS

tlink [**–chnprvX**] [**–d***name*] [**–x***name*] *source target* [*path*...]

DESCRIPTION

Tlink creates a directory tree rooted at *target* identical to the directory tree rooted at *source*, populating the directories in the target tree with symbolic links to corresponding files under the source tree. If optional *paths* are supplied after *source* and *target*, only the sub-trees named by concatenating each path with *source* are linked under *target*.

The **–c** (clean) option causes *tlink* to walk the target tree removing any directory which lacks an isomorph in the source tree, any symbolic link which does not name its non-directory isomorph in the source tree, and any file which is not a directory or a link.

The **–d** option symbolically links a directory in the source tree into the corresponding place in the target tree. The link's pathname must match the regular expression named. Regular expressions are as described in *regcmp*(3X). When used with **–c**, this option prevents *tlink* from cleaning symbolic links to source directories.

The **–h** option creates hard rather than symbolic links, to conserve inodes and disk blocks in a filesystem. A hard-linked tree has the drawback that a file linked in it may become stale (i.e. diverge from its prototype source node) if its source is unlinked and recreated, whereas a symbolic link to the source will always denote the same source name. This option fails if source and target are directories in different filesystems.

The **–n** option causes *tlink* to operate without actually constructing a target tree. With this option, *tlink* will traverse the source tree, formulating pathnames, changing current directory, and calling *stat*(2) on source files.

The **–p** (prune) option causes tlink to remove dangling symbolic links from the target tree. Prior *tlink* invocations may have created links to source files that no longer exist; **tlink –p** removes these dead leaves.

To create relative rather than absolute symbolic links, use the **–r** option. *Tlink* relates targets to sources by computing the path up from each target to the closest ancestor directory common to source and target, and appending the source path down from this ancestor.

The **–v** (verbose) option prints the name of each directory and symbolic link created (or removed with the **–c** option). If a non-directory file exists in the target tree and its source file is a directory, **tlink –v** prints the target's pathname and ''Not a directory.'' If a directory in the target tree has a non-directory source, verbose *tlink* prints the target's pathname and ''Is a directory.'' If a symbolic link in the target tree names no existent file, then *tlink* will attempt to unlink the stale link. Upon successful removal, verbose *tlink* will print the target's pathname and "No such file or directory."

The **–x** option adds regular expressions describing filenames to be excluded from the tree walk to a list. The list's initial contents are:

```
^\.{1,2}$
^RCS$
^.*,v$
```

If a regular expression contains slashes, then *tlink* matches source pathnames rather than filenames against the expression. Specifying –X eliminates all but the first of these expressions from the exclusion list.

AUTHOR

Brendan Eich, 01/14/87

SEE ALSO

stat(2), regexp(3X).

NAME

tobw – convert a color image to black and white

SYNOPSIS

tobw *colorimage bwimage*

DESCRIPTION

tobw converts a stored color image to black and white and saves it. The color file must already exist as *colorimage;* the black and white file will be named *bwimage*. Both files use the file access methodology described in Appendix H of the Users' Guide, "Using the Image Library".

NAME

toolchest, windowchest, demochest – 4Sight utility and windowing toolchests

SYNOPSIS

toolchest
windowchest
demochest

DESCRIPTION

toolchest and *windowchest* create the Tools and Windows menus that appear when 4Sight is loaded at login time. */usr/sbin/toolchest* and */usr/sbin/windowchest* are executable PostScript files (*psh* scripts) containing the definitions of the Tools and Windows menus. Copies of these file may be placed in the user's home directory and modified; they will override the default copies when invoked.

demochest creates the Demos menu, which contains a variety of NeWS and GL demo programs. *demochest* also resides in */usr/sbin*, and can be modified in the same manner as *toolchest* and *windowchest*. *demochest* is not invoked at startup, but this can be accomplished by adding the following PostScript code to the file *$HOME/user.ps*:

```
RestartActions [
      { (demochest) forkunix }
] def
```

SEE ALSO

journalchest(1W)

4Sight User's Guide, Section 3, Chapter 4.

NAME

top – display processes having highest CPU usage

SYNOPSIS

top [**–i** interval]

DESCRIPTION

This command displays a sorted list of processes which are using some portion of the available CPU cycles on a machine. The display is updated every interval.

The following fields are displayed in order for each process: user name, process state flags, process ID, process group ID, CPU usage, processor currently executing the process, process priority, process size (in pages), resident set size (in pages), amount of CPU time used by the process, and the process name.

Two header lines are displayed. The first gives the machine name, the release and build date information, the processor type, the current time and the number of active processes. The next line is a header containing the name of each field highlighted.

If run from a job control shell, job control commands are fully supported. Typing a ^-L causes *top* to redraw the screen, and an interrupt causes the program to exit.

OPTIONS

The following option is supported:

–i *interval*	This option sets the update interval used; by default this is 5 seconds.

AUTHOR

J. M. Barton, Silicon Graphics, Inc.

NAME

touch — update access and modification times of a file

SYNOPSIS

touch [**-amc**] [mmddhhmm[yy]] files

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The **−a** and **−m** options cause touch to update only the access or modification times respectively (default is **−am**). The **−c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1).
utime(2) in the *Programmer's Reference Manual.*

BUGS

Trying to *touch* a file whose name is only numbers, e.g. **touch 1234**, will fail since *touch* tries to interpret the number as the time rather than a file name. To *touch* such a file, use **touch ./1234**.

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

tput [–Ttype] capname [parms ...]

tput [–Ttype] **init**

tput [–Ttype] **reset**

tput [–Ttype] **longname**

DESCRIPTION

tput uses the *terminfo*(4) database to make the values of terminal-dependent capabilities and information available to the shell (see *sh*(1)), to initialize or reset the terminal, or return the long name of the requested terminal type. *tput* outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, *tput* simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code ($?, see *sh*(1)) to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS** below.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*(4).

–T*type*	indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable **TERM**. If –T is specified, then the shell variables **LINES** and **COLUMNS**.
capname	indicates the attribute from the *terminfo*(4) database.
parms	If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.
init	If the *terminfo*(4) database is present and an entry for the user's terminal exists (see –T*type*, above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (e.g., newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **–T***type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the *terminfo*(4) database (see *term*(5)).

EXAMPLES

tput init Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command should be included in everyone's .profile after the environmental variable **TERM** has been exported, as illustrated on the *profile*(4) manual page.

tput –T5620 reset Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM**.

tput cup 0 0 Send the sequence to move the cursor to row 0, column 0 (the upper left corner of the screen, usually known as the "home" cursor position).

tput clear Echo the clear-screen sequence for the current terminal.

tput cols Print the number of columns for the current terminal.

tput -T450 cols Print the number of columns for the 450 terminal.

bold=‘tput smso‘

offbold=‘tput rmso‘ Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt:
echo "${bold}Please type in your name: ${offbold}\c"

tput hc Set exit code to indicate if the current terminal is a hardcopy terminal.

tput cup 23 4 Send the sequence to move the cursor to row 23, column 4.

tput longname	Print the long name from the *terminfo*(4) database for the type of terminal specified in the environmental variable **TERM**.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/curses.h	*curses*(3X) header file
/usr/include/term.h	*terminfo*(4) header file
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of *terminfo*(4)

SEE ALSO

stty (1), tabs (1).
profile(4), terminfo(4) in the *Programmer's Reference Manual.*
Chapter 9 of the *Programmer's Guide*.

EXIT CODES

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of −1 means that *capname* is not defined for this terminal *type*.

Any other exit code indicates an error; see **DIAGNOSTICS**, below.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

exit code	error message
0	−1 (*capname* is a numeric variable that is not specified in the *terminfo*(4) database for this terminal type, e.g. **tput −T450 lines** and **tput −T2621 xmc**)
1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) database
4	unknown *terminfo*(4) capability *capname*

NAME

tr – translate characters

SYNOPSIS

tr [**–cds**] [string1 [string2]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **–cds** may be used:

–c	Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
–d	Deletes all input characters in *string1*.
–s	Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a–z]	Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.
[a**n*]	Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr –cs "[A–Z][a–z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1).
ascii(5) in the *Programmer's Reference Manual.*

BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

transfermanager – provide a visual interface for selecting entries in the workspace transfer menu

SYNOPSIS

transfermanager

DESCRIPTION

transfermanager is a WorkSpace utility that is accessible through the System Chest or the IRIX command line. *transfermanager* provides a flexible means for customizing the WorkSpace and Directory View popup menus in order to integrate common device and/or processing interfaces. A number of operations are supported; other operations can be added; each piece of functionality is provided by a distinct *transferdevice*.

transferdevices are specialized shell scripts that implement one or more possible menu actions and are recognized by the standard WorkSpace file typing rules.

The main *transfermanager* window contains an iconic representation of all available *transferdevices*. The set of available *transferdevices* are thosc devices found in */etc/transferDevices* (that are flagged as usable in their prototype form) and any user modified versions found in $HOME/.workspace/localTransferLinks. The operation of the *transfermanager* is controlled by the six buttons in the lower section of the window.

BUTTONS

The name of the leftmost button is dependent on the login of the current user. It appears under the label "Assign current devices for:"

user is used to assign the currently selected device or devices as current. When a device is selected as current, its icon will appear backed by a green placard in the view. All of the menu entries that it implements will be added to the transfer submenu of WorkSpace.

Add allows the user to create a customized version of one of the prototype devices. A device can be customized to work over the network on a particular remote machine. Add presents the user with a dialog that prompts for the device type and remote machine name.

Delete removes the selected devices if they have been added by the user.

Open presents the user with a window describing the functionality of the selected *transferdevices*. If the device is one of the user has added, he is given the opportunity to edit the remote machine name.

Quit	closes down the *transfermanager*.
HELP	is used to access the visual help system.

FILES

/etc/transferDevice/
$HOME/.workspace/localTransferLinks/
$HOME/.workspace/currentTransferDevice
$HOME/.workspace/.tmLock

SEE ALSO

workspace (1G)
Programming the IRIS WorkSpace

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

true has exit status zero, *false* nonzero.

NAME

tset – terminal dependent initialization

SYNOPSIS

tset [options]

DESCRIPTION

Tset causes terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It first determines the *type* of terminal involved, names for which are specified by the */usr/lib/terminfo* data base, and then does necessary initializations and mode settings. In the case where no argument types are specified, *tset* simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual concerns itself with type initialization, done typically once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script ".profile" (for *sh*(1) users) or ".login" (for *csh*(1) users), it is desirable to give information about the types of terminal usually used, for terminals which are connected to the computer through a modem. These ports are initially identified as being *dialup* or *plugboard* or *arpanet* etc. To specify what terminal type is usually used on these ports, **–m** is followed by the appropriate port type identifier, an optional baud-rate specification, and the terminal type to be used if the mapping conditions are satisfied. If more than one mapping is specified, the first applicable mapping prevails. A missing type identifier matches all identifiers.

Baud rates are specified as with *stty*(1), and are compared with the speed of the diagnostic output (which is almost always the control terminal). The baud rate test may be any combination of: >, =, <, @, and !; @ is a synonym for = and ! inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **–m** within ´´ characters; users of *csh*(1) must also put a "\" before any "!" used here.

Thus

```
tset –m ´dialup>300:adm3a´ –m dialup:dw2
```

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (i.e., at 300 baud or less). If the *type* above begins with a question mark, the user is asked if the user really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. For other ports the port type will be taken from the **/etc/ttytype** file or a final, default *type* option may be given on the command line not preceded by a **–m**. A ttytype may be preceded with a question mark in **/etc/ttytype** for prompting (this is an enhancement over standard *tset*).

It is often desirable to return the terminal type, as specified by the **–m** options, and information about the terminal to a shell's environment. This can be done using the –s option; using the Bourne shell, *sh*(1):

```
eval `tset –s options ...
```

or using the C shell, *csh*(1):

```
tset /-s options ... > tset$$
source tset$$
rm tset$$
```

These commands cause *tset* to generate as output a sequence of shell commands which place the variables TERM and TERMCAP in the environment; see *environ*(4).

Once the terminal type is known, *tset* engages in terminal mode setting. This normally involves sending an initialization sequence to the terminal and setting the single character erase (and optionally the line-kill (full line erase)) characters.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ("#" on standard systems), the erase character is changed to a ^H (backspace).

Other options are:

- **–e** set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually ^H.
- **–k** is similar to **–e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons); ^U is the preferred setting. No kill processing is done if **–k** is not specified.
- **–I** suppresses outputting terminal initialization strings.
- **–Q** suppresses printing the "Erase set to" and "Kill set to" messages.
- **–S** Outputs just the strings to be assigned to TERM and TERMCAP rather than commands for a shell.

EXAMPLES

A typical *csh* ".login" file using *tset* would be:

```
set noglob
set tmp = (tset /-e /-S /-r /-d?h19)
setenv TERM "$tmp[1]"
unset tmp noglob
```

This ".login" sets the environment variable TERM for the user's current terminal according to the file */etc/ttytype* . If the terminal line is a dialup line, the user is prompted for the proper terminal type.

ENVIRONMENT

The –s option uses SHELL to determine the syntax of the output.

FILES

/etc/ttytype	terminal id to type map database
/usr/lib/terminfo	terminal capability database

SEE ALSO

csh(1), reset(1), sh(1), stty(1), environ(4), ttytype(4), termcap(4).

BUGS

Should be merged with *stty*(1).

It could well be argued that the shell should be responsible for insuring that the terminal remains in a sane state; this would eliminate the need for this program.

NOTES

For compatibility with earlier versions of *tset*, a number of flags are accepted whose use is discouraged:

–d type	equivalent to –m dialup:type
–p type	equivalent to –m plugboard:type
–a type	equivalent to –m arpanet:type
–E c	Sets the erase character to *c* only if the terminal can backspace.
–	prints the terminal type on the standard output
–r	prints the terminal type on the diagnostic output.

AUTHOR

Eric Allman

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

The *tsort* command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NAME

tty – get the name of the terminal

SYNOPSIS

tty [**–l**] [**–s**]

DESCRIPTION

tty prints the path name of the user's terminal.

–l	prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.
–s	inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES

2	if invalid options were specified,
0	if standard input is a terminal,
1	otherwise.

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and –l is specified.
"not a tty" if the standard input is not a terminal and –s is not specified.

NAME

twm - Tab Window Manager for the X Window System

SYNTAX

twm [-display *dpy*] [-s] [-f *initfile*] [-v]

DESCRIPTION

Twm is a window manager for the X Window System. It provides titlebars, shaped windows, several forms of icon management, user-defined macro functions, click-to-type and pointer-driven keyboard focus, and user-specified key and pointer button bindings.

This program is usually started by the user's session manager or startup script. When used from *xdm(1)* or *xinit(1)* without a session manager, *twm* is frequently executed in the foreground as the last client. When run this way, exiting *twm* causes the session to be terminated (i.e. logged out).

By default, application windows are surrounded by a ''frame'' with a titlebar at the top and a special border around the window. The titlebar contains the window's name, a rectangle that is lit when the window is receiving keyboard input, and function boxes known as ''titlebuttons'' at the left and right edges of the titlebar.

Pressing pointer Button1 (usually the left-most button unless it has been changed with *xmodmap*) on a titlebutton will invoke the function associated with the button. In the default interface, windows are iconified by clicking (pressing and then immediately releasing) the left titlebutton (which looks like a small X). Conversely, windows are deiconified by clicking in the associated icon or entry in the icon manager (see description of the variable **ShowIconManager** and of the function **f.showiconmgr**).

Windows are resized by pressing the right titlebutton (which resembles group of nested squares), dragging the pointer over edge that is to be moved, and releasing the pointer when the outline of the window is the desired size. Similarly, windows are moved by pressing in the title or highlight region, dragging a window outline to the new location, and then releasing when the outline is in the desired position. Just clicking in the title or highlight region raises the window without moving it.

When new windows are created, *twm* will honor any size and location information requested by the user (usually through *-geometry* command line argument or resources for the individual applications). Otherwise, an outline of the window's default size, its titlebar, and lines dividing the window into a 3x3 grid that track the pointer are displayed. Clicking pointer Button1 will position the window at the current position and give it the default size. Pressing pointer Button2 (usually the middle pointer button) and dragging the outline will give the window its current position but allow the sides to be resized as described above. Clicking pointer Button3 (usually the right pointer button) will give the window its current position but attempt to make it long enough to touch the bottom the screen.

OPTIONS

Twm accepts the following command line options:

—display *dpy*
: This option specifies the X server to use.

—s
: This option indicates that only the default screen (as specified by **–display** or by the **DISPLAY** environment variable) should be managed. By default, *twm* will attempt to manage all screens on the display.

—f *filename*
: This option specifies the name of the startup file to use. By default, *twm* will look in the user's home directory for files named

—v
: This option indicates that *twm* should print error messages whenever an unexpected X Error event is received. This can be useful when debugging applications but can be distracting in regular use.

CUSTOMIZATION

Much of *twm*'s appearance and behavior can be controlled by providing a startup file in one of the following locations (searched in order for each screen being managed when *twm* begins):

$HOME/.twmrc.*screennumber*
: The *screennumber* is a small positive number (e.g. 0, 1, etc.) representing the screen number (e.g. the last number in the DISPLAY environment variable *host:displaynum.screennum*) that would be used to contact that screen of the display. This is intended for displays with multiple screens of differing visual types.

$HOME/.twmrc
: This is the usual name for an individual user's startup file.

/usr/lib/X11/twm/system.twmrc
: If neither of the preceding files are found, *twm* will look in this file for a default configuration. This is often tailored by the site administrator to provide convenient menus or familiar bindings for novice users.

If no startup files are found, *twm* will use the built-in defaults described above. The only resource used by *twm* is *bitmapFilePath* for a colon-separated list of directories to search when looking for bitmap files (for more information, see the *Athena Widgets* manual and *xrdb(1)*).

Twm startup files are logically broken up into three types of specifications: *Variables*, *Bindings*, *Menus*. The *Variables* section must come first and is used to describe the fonts, colors, cursors, border widths, icon and window placement, highlighting, autoraising, layout of titles, warping, use of the icon manager. The *Bindings* section usually comes second and is used to specify the functions that should be to be invoked when keyboard and pointer buttons are pressed in windows, icons, titles, and frames. The *Menus* section gives any user-defined menus (containing functions to be invoked or commands to be executed).

Variable names and keywords are case-insensitive. Strings must be surrounded by double quote characters (e.g. "blue") and are case-sensitive. A pound sign (#) outside of a string causes the remainder of the line in which the character appears to be treated as a comment.

VARIABLES

Many of the aspects of *twm*'s user interface are controlled by variables that may be set in the user's startup file. Some of the options are enabled or disabled simply by the presence of a particular keyword. Other options require keywords, numbers, strings, or lists of all of these.

Lists are surrounded by braces and are usually separated by whitespace or a newline. For example:

```
AutoRaise { "emacs" "XTerm" "Xmh" }
```

or

```
AutoRaise
{
        "emacs"
        "XTerm"
        "Xmh"
}
```

When a variable containing a list of strings representing windows is searched (e.g. to determine whether or not to enable autoraise as shown above), a string is considered to match a window if it is a case-sensitive prefix for the window's name name (given by the WM_NAME window property), resource name or class name (both given by the WM_CLASS window property). The preceding example would enable autoraise on windows named "emacs" as well as any *xterm* (since they are of class "XTerm") or xmh windows (which are of class "Xmh").

String arguments that are interpreted as filenames (see the **Pixmaps**, **Cursors**, and **IconDirectory** below) will prepend the user's directory (specified by the **HOME** environment variable) if the first character is a tilde (~). If, instead, the first character is a colon (:), the name is assumed to refer to one of the internal bitmaps that are used to create the default titlebars symbols: **:xlogo** or **:iconify** (both refer to the X used for the iconify button), **:resize** (the nested squares used by the resize button), and **:question** (the question mark used for non-existent bitmap files).

The following variables may be specified at the top of a *twm* startup file. Lists of Window name prefix strings are indicated by *win-list*. Optional arguments are shown in square brackets:

AutoRaise { *win-list* }

This variable specifies a list of windows that should automatically be raised whenever the pointer enters the window. This action can be interactively enabled or disabled on individual windows using the function **f.autoraise**.

AutoRelativeResize

This variable indicates that dragging out a window size (either when initially sizing the window with pointer Button2 or when resizing it) should not wait until the pointer has crossed the window edges. Instead, moving the pointer automatically causes the nearest edge or edges to move by the same amount. This allows allows the resizing windows that extend off the edge of the screen. If the pointer is in the center of the window, or if the resize is begun by pressing a titlebutton, *twm* will still wait for the pointer to cross a window edge (to prevent accidents). This option is particularly useful for people who like the press-drag-release method of sweeping out window sizes.

BorderColor *string* [{ *wincolorlist* }]

This variable specifies the default color of the border to be placed around all non-iconified windows, and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* specifies a list of window and color name pairs for specifying particular border colors for different types of windows. For example:

```
BorderColor "gray50"
{
        "XTerm"         "red"
        "xmh"   "green"
}
```

The default is "black".

BorderTileBackground *string* [{ *wincolorlist* }]

This variable specifies the default background color in the gray pattern used in unhighlighted borders (only if **NoHighlight** hasn't been set), and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* allows per-window colors to be specified. The default is "black".

BorderTileForeground *string* [{ *wincolorlist* }]
: This variable specifies the default foreground color in the gray pattern used in unhighlighted borders (only if **NoHighlight** hasn't been set), and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* allows per-window colors to be specified. The default is "white".

BorderWidth *pixels*
: This variable specifies the width in pixels of the border surrounding all client window frames if **ClientBorderWidth** has not been specified. This value is also used to set the border size of windows created by *twm* (such as the icon manager). The default is 2.

ButtonIndent *pixels*
: This variable specifies the amount by which titlebuttons should be indented on all sides. Positive values cause the buttons to be smaller than the window text and highlight area so that they stand out. Setting this and the **TitleButtonBorderWidth** variables to 0 makes titlebuttons be as tall and wide as possible. The default is 1.

ClientBorderWidth
: This variable indicates that border width of a window's frame should be set to the initial border width of the window, rather than to the value of **BorderWidth**.

Color { *colors-list* }
: This variable specifies a list of color assignments to be made if the default display is capable of displaying more than simple black and white. The *colors-list* is made up of the following color variables and their values: **DefaultBackground**, **DefaultForeground**, **MenuBackground**, **MenuForeground**, **MenuTitleBackground**, **MenuTitleForeground**, and **MenuShadowColor**. The following color variables may also be given a list of window and color name pairs to allow per-window colors to be specified (see **BorderColor** for details): **BorderColor**, **IconManagerHighlight**, **BorderTitleBackground**, **BorderTitleForeground**, **TitleBackground**, **TitleForeground**, **IconBackground**, **IconForeground**, **IconBorderColor**, **IconManagerBackground**, and **IconManagerForeground**.

For example:

```
Color
{
        MenuBackground          "gray50"
        MenuForeground          "blue"
        BorderColor             "red" { "XTerm" "yellow" }
        TitleForeground         "yellow"
        TitleBackground         "blue"
}
```

All of these color variables may also be specified for the **Monochrome** variable, allowing the same initialization file to be used on both color and monochrome displays.

ConstrainedMoveTime *milliseconds*

This variable specifies the length of time between button clicks needed to begin a constrained move operation. Double clicking within this amount of time when invoking **f.move** will cause the window only be moved in a horizontal or vertical direction. Setting this value to 0 will disable constrained moves. The default is 400 milliseconds.

Cursors { *cursor-list* }

This variable specifies the glyphs that *twm* should use for various pointer cursors. Each cursor may be defined either from the **cursor** font or from two bitmap files. Shapes from the **cursor** font may be specified directly as:

cursorname "*string*"

where *cursorname* is one of the cursor names listed below, and *string* is the name of a glyph as found in the file /usr/include/X11/cursorfont.h (without the ''XC_'' prefix). If the cursor is to be defined from bitmap files, the following syntax is used instead:

cursorname "*image*" "*mask*"

The *image* and *mask* strings specify the names of files containing the glyph image and mask in *bitmap(1)* form. The bitmap files are located in the same manner as icon bitmap files. The following example shows the default cursor definitions:

```
Cursors
{
        Frame           "top_left_arrow"
        Title           "top_left_arrow"
```

```
                    Icon            "top_left_arrow"
                    IconMgr"top_left_arrow"
                    Move            "fleur"
                    Resize          "fleur"
                    Menu            "sb_left_arrow"
                    Button          "hand2"
                    Wait            "watch"
                    Select          "dot"
                    Destroy "pirate"
            }
```

DecorateTransients
This variable indicates that transient windows (those containing a WM_TRANSIENT_FOR property) should have titlebars. By default, transients are not reparented.

DefaultBackground *string*
This variable specifies the background color to be used for sizing and information windows. The default is "white".

DefaultForeground *string*
This variable specifies the foreground color to be used for sizing and information windows. The default is "black".

DontIconifyByUnmapping { *win-list* }
This variable specifies a list of windows that should not be iconified by simply unmapping the window (as would be the case if **IconifyByUnmapping** had been set). This is frequently used to force some windows to be treated as icons while other windows are handled by the icon manager.

DontMoveOff
This variable indicates that windows should not be allowed to be moved off the screen. It can be overridden by the **f.forcemove** function.

DontSqueezeTitle [{ *win-list* }]
This variable indicates that titlebars should not be squeezed to their minimum size as described under **SqueezeTitle** below. If the optional window list is supplied, only those windows will be prevented from being squeezed.

ForceIcons
This variable indicates that icon pixmaps specified in the **Icons** variable should override any client-supplied pixmaps.

FramePadding *pixels*
This variable specifies the distance between the titlebar decorations (the button and text) and the window frame. The default is 2 pixels.

IconBackground *string* [{ *win-list* }]
This variable specifies the background color of icons, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "white".

IconBorderColor *string* [{ *win-list* }]
This variable specifies the color of the border used for icon windows, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconBorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding icon windows. The default is 2.

IconDirectory *string*
This variable specifies the directory that should be searched if if a bitmap file cannot be found in any of the directories in the **bitmapFilePath** resource.

IconFont *string*
This variable specifies the font to be used to display icon names within icons. The default is "8x13".

IconForeground *string* [{ *win-list* }]
This variable specifies the foreground color to be used when displaying icons, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconifyByUnmapping [{ *win-list* }]
This variable indicates that windows should be iconified by being unmapped without trying to map any icons. This assumes that the user is will remap the window through the icon manager, the **f.warpto** function, or the *TwmWindows* menu. If the optional *win-list* is provided, only those windows will be iconified by simply unmapping. Windows that have both this and the

IconManagerDontShow options set may not be accessible if no binding to the *TwmWindows* menu is set in the user's startup file.

IconManagerBackground *string* [{ *win-list* }]
: This variable specifies the background color to use for icon manager entries, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "white".

IconManagerDontShow [{ *win-list* }]
: This variable indicates that the icon manager should not display any windows. If the optional *win-list* is given, only those windows will not be displayed. This variable is used to prevent windows that are rarely iconified (such as *xclock* or *xload*) from taking up space in the icon manager.

IconManagerFont *string*
: This variable specifies the font to be used when displaying icon manager entries. The default is "8x13".

IconManagerForeground *string* [{ *win-list* }]
: This variable specifies the foreground color to be used when displaying icon manager entries, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconManagerGeometry *string* [*columns*]
: This variable specifies the geometry of the icon manager window. The *string* argument is standard geometry specification that indicates the initial full size of the icon manager. The icon manager window is then broken into *columns* pieces and scaled according to the number of entries in the icon manager. Extra entries are wrapped to form additional rows. The default number of columns is 1.

IconManagerHighlight *string* [{ *win-list* }]
: This variable specifies the border color to be used when highlighting the icon manager entry that currently has the focus, and can only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconManagers { *iconmgr-list* }

This variable specifies a list of icon managers to create. Each item in the *iconmgr-list* has the following format:

"*winname*" ["*iconname*"] "*geometry*" *columns*

where *winname* is the name of the windows that should be put into this icon manager, *iconname* is the name of that icon manager window's icon, *geometry* is a standard geometry specification, and *columns* is the number of columns in this icon manager as described in **IconManagerGeometry**. For example:

```
IconManagers
{
        "XTerm"          "=300x5+800+5" 5
        "myhost"         "=400x5+100+5" 2
}
```

Clients whose name or class is ``XTerm'' will have an entry created in the ``XTerm'' icon manager. Clients whose name was ``myhost'' would be put into the ``myhost'' icon manager.

IconManagerShow { *win-list* }

This variable specifies a list of windows that should appear in the icon manager. When used in conjunction with the **IconManager-DontShow** variable, only the windows in this list will be shown in the icon manager.

IconRegion *geomstring vgrav hgrav gridwidth gridheight*

This variable specifies an area on the root window in which icons are placed if no specific icon location is provided by the client. The *geomstring* is a quoted string containing a standard geometry specification. If more than one **IconRegion** lines are given, icons will be put into the succeeding icon regions when the first is full. The *vgrav* argument should be either **North** or **South** and control and is used to control whether icons are first filled in from the top or bottom of the icon region. Similarly, the *hgrav* argument should be either **East** or **West** and is used to control whether icons should be filled in from left from the right. Icons are laid out within the region in a grid with cells *gridwidth* pixels wide and *gridheight* pixels high.

Icons { *win-list* }

This variable specifies a list of window names and the bitmap filenames that should be used as their icons. For example:

```
Icons
{
        "XTerm"         "xterm.icon"
        "xfd"           "xfd_icon"
}
```

Windows that match "XTerm" and would not be iconified by unmapping, and would try to use the icon bitmap in the file "xterm.icon". If **ForceIcons** is specified, this bitmap will be used even if the client has requested its own icon pixmap.

InterpolateMenuColors

This variable indicates that menu entry colors should be interpolated between entry specified colors. In the example below:

```
Menu "mymenu"
{
        "Title"         ("black":"red")         f.title
        "entry1"                                f.nop
        "entry2"                                f.nop
        "entry3"        ("white":"green")       f.nop
        "entry4"                                f.nop
        "entry5"        ("red":"white")         f.nop
}
```

the foreground colors for "entry1" and "entry2" will be interpolated between black and white, and the background colors between red and green. Similarly, the foreground for "entry4" will be half-way between white and red, and the background will be half-way between green and white.

MakeTitle { *win-list* }

This variable specifies a list of windows on which a titlebar should be placed and is used to request titles on specific windows when **NoTitle** has been set.

MaxWindowSize *string*

This variable specifies a geometry in which the width and height give the maximum size for a given window. This is typically used to restrict windows to the size of the screen. The default is "30000x30000".

MenuBackground *string*

This variable specifies the background color used for menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

MenuFont *string*
This variable specifies the font to use when displaying menus. The default is "8x13".

MenuForeground *string*
This variable specifies the foreground color used for menus, and can only be specified inside of a Color or Monochrome list. The default is "black".

MenuShadowColor *string*
This variable specifies the color of the shadow behind pull-down menus and can only be specified inside of a Color or Monochrome list. The default is "black".

MenuTitleBackground *string*
This variable specifies the background color for **f.title** entries in menus, and can only be specified inside of a Color or Monochrome list. The default is "white".

MenuTitleForeground *string*
This variable specifies the foreground color for **f.title** entries in menus and can only be specified inside of a Color or Monochrome list. The default is "black".

Monochrome { *colors* }
This variable specifies a list of color assignments that should be made if the screen has a depth of 1. See the description of Colors.

MoveDelta *pixels*
This variable specifies the number of pixels the pointer must move before the **f.move** function starts working. Also see the **f.deltastop** function. The default is zero pixels.

NoBackingStore
This variable indicates that *twm*'s menus should not request backing store to minimize repainting of menus. This is typically used with servers that can repaint faster than they can handle backing store.

NoCaseSensitive
This variable indicates that case should be ignored when sorting icon names in an icon manager. This option is typically used with applications that capitalize the first letter of their icon name.

NoDefaults
This variable indicates that *twm* should not supply the default titlebuttons and bindings. This option should only be used if the startup file contains a completely new set of bindings and

definitions.

NoGrabServer
This variable indicates that *twm* should not grab the server when popping up menus and moving opaque windows.

NoHighlight [{ *win-list* }]
This variable indicates that borders should not be highlighted to track the location of the pointer. If the optional *win-list* is given, highlighting will only be disabled for those windows. When the border is highlighted, it will be drawn in the current **Border-Color**. When the border is not highlighted, it will be stippled with an gray pattern using the current **BorderTileForeground** and **BorderTileBackground** colors.

NoIconManagers
This variable indicates that no icon manager should be created.

NoMenuShadows
This variable indicates that menus should not have drop shadows drawn behind them. This is typically used with slower servers since it speeds up menu drawing at the expense of making the menu slightly harder to read.

NoRaiseOnDeiconify
This variable indicates that windows that are deiconified should not be raised.

NoRaiseOnMove
This variable indicates that windows should not be raised when moved. This is typically used to allow windows to slide underneath each other.

NoRaiseOnResize
This variable indicates that windows should not be raised when resized. This is typically used to allow windows to be resized underneath each other.

NoRaiseOnWarp
This variable indicates that windows should not be raised when the pointer is warped into them with the **f.warpto** function. If this option is set, warping to an occluded window may result in the pointer ending up in the occluding window instead the desired window (which causes unexpected behavior with **f.warpring**).

NoSaveUnders
This variable indicates that menus should not request save-unders to minimize window repainting following menu selection. It is typically used with displays that can repaint faster than they can

handle save-unders.

NoTitle [{ *win-list* }]
This variable indicates that windows should not have titlebars. If the optional *win-list* is given, only those windows will not have titlebars. **MakeTitle** may be used with this option to force titlebars to be put on specific windows.

NoTitleFocus
This variable indicates that *twm* should not set keyboard input focus to each window as it is entered. Normally, *twm* sets the focus so that focus and key events from the titlebar and icon managers are delivered to the application. If the pointer is moved quickly and *twm* is slow to respond, input can be directed to the old window instead of the new. This option is typically used to prevent this ''input lag'' and to work around bugs in older applications that have problems with focus events.

NoTitleHighlight [{ *win-list* }]
This variable indicates that the highlight area of the titlebar, which is used to indicate the window that currently has the input focus, should not be displayed. If the optional *win-list* is given, only those windows will not have highlight areas. This and the **SqueezeTitle** options can be set to substantially reduce the amount of screen space required by titlebars.

OpaqueMove
This variable indicates that the **f.move** function should actually move the window instead of just an outline so that the user can immediately see what the window will look like in the new position. This option is typically used on fast displays (particularly if **NoGrabServer** is set).

Pixmaps { *pixmaps* }
This variable specifies a list of pixmaps that define the appearance of various images. Each entry is a keyword indicating the pixmap to set, followed by a string giving the name of the bitmap file. The following pixmaps may be specified:

```
Pixmaps
{
        TitleHighlight    "gray1"
}
```

The default for *TitleHighlight* is to use an even stipple pattern.

RandomPlacement

This variable indicates that windows with no specified geometry should should be placed in a pseudo-random location instead of having the user drag out an outline.

ResizeFont *string*

This variable specifies the font to be used for in the dimensions window when resizing windows. The default is "fixed".

RestartPreviousState

This variable indicates that *twm* should attempt to use the WM_STATE property on client windows to tell which windows should be iconified and which should be left visible. This is typically used to make try to regenerate the state that the screen was in before the previous window manager was shutdown.

ShowIconManager

This variable indicates that the icon manager window should be displayed when *twm* is started. It can always be brought up using the **f.showiconmgr** function.

SortIconManager

This variable indicates that entries in the icon manager should be sorted alphabetically rather than by simply appending new windows to the end.

SqueezeTitle [{ *squeeze-list* }]

This variable indicates that *twm* should attempt to use the SHAPE extension to make titlebars occupy only as much screen space as they need, rather than extending all the way across the top of the window. The optional *squeeze-list* may be used to control the location of the squeezed titlebar along the top of the window. It contains entries of the form:

"*name*" *justification* *num* *denom*

where *name* is a window name, *justification* is either **left**, **center**, or **right**, and *num* and *denom* are numbers specifying a ratio giving the relative position about which the titlebar is justified. The ratio is measured from left to right if the numerator is positive, and right to left if negative. A denominator of 0 indicates that the numerator should be measured in pixels. For convenience, the ratio 0/0 is the same as 1/2 for **center** and -1/1 for **right**. For example:

```
SqueezeTitle
{
        "XTerm"     left        0    0
        "xterm1"    left        1    3
        "xterm2"    left        2    3
        "oclock" center     0    0
        "emacs" right       0    0
}
```

The **DontSqueezeTitle** list can be used to turn off squeezing on certain titles.

StartIconified [{ *win-list* }]
This variable indicates that client windows should initially be left as icons until explicitly deiconified by the user. If the optional *win-list* is given, only those windows will be started iconic. This is useful for programs that do not support an *-iconic* command line option or resource.

TitleBackground *string* [{ *win-list* }]
This variable specifies the background color used in titlebars, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. The default is "white".

TitleButtonBorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding titlebuttons. This is typically set to 0 to allow titlebuttons to take up as much space as possible and to not have a border. The default is 1.

TitleFont *string*
This variable specifies the font to used for displaying window names in titlebars. The default is "8x13".

TitleForeground *string* [{ *win-list* }]
This variable specifies the foreground color used in titlebars, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. The default is "black".

TitlePadding *pixels*
This variable specifies the distance between the various buttons, text, and highlight areas in the titlebar. The default is 8 pixels.

UnknownIcon *string*
: This variable specifies the filename of a bitmap file to be used as the default icon. This bitmap will be used as the icon of all clients which do not provide an icon bitmap and are not listed in the **Icons** list.

UsePPosition *string*
: This variable specifies whether or not *twm* should honor program-requested locations (given by the **PPosition** flag in the WM_NORMAL_HINTS property) in the absence of a user-specified position. The argument *string* may have one of three values: **"off"** (the default) indicating that *twm* should ignore the program-supplied position, **"on"** indicating that the position should be used, and **"non-zero"** indicating that the position should used if it is other than (0,0). The latter option is for working around a bug in older toolkits.

WarpCursor [{ *win-list* }]
: This variable indicates that the pointer should be warped into windows when they are deiconified. If the optional *win-list* is given, the pointer will only be warped when those windows are deiconified.

WindowRing { *win-list* }
: This variable specifies a list of windows along which the **f.warpring** function cycles.

WarpUnmapped
: This variable indicates that that the **f.warpto** function should deiconify any iconified windows it encounters. This is typically used to make a key binding that will pop a particular window (such as *xmh*), no matter where it is. The default is for **f.warpto** to ignore iconified windows.

XorValue *number*
: This variable specifies the value to use when drawing window outlines for moving and resizing. This should be set to a value that will result in a variety of of distinguishable colors when exclusive-or'ed with the contents of the user's typical screen. Setting this variable to 1 often gives nice results if adjacent colors in the default colormap are distinct. By default, *twm* will attempt to cause temporary lines to appear at the opposite end of the colormap from the graphics.

Zoom [*count*]

This variable indicates that outlines suggesting movement of a window to and from its iconified state should be displayed whenever a window is iconified or deiconified. The optional *count* argument specifies the number of outlines to be drawn. The default count is 8.

The following variables must be set after the fonts have been assigned, so it is usually best to put them at the end of the variables or beginning of the bindings sections:

DefaultFunction *function*

This variable specifies the function to be executed when a key or button event is received for which no binding is provided. This is typically bound to **f.nop**, **f.beep**, or a menu containing window operations.

WindowFunction *function*

This variable specifies the function to execute when a window is selected from the **TwmWindows** menu. If this variable is not set, the window will be deiconified and raised.

BINDINGS

After the desired variables have been set, functions may be attached titlebuttons and key and pointer buttons. Titlebuttons may be added from the left or right side and appear in the titlebar from left-to-right according to the order in which they are specified. Key and pointer button bindings may be given in any order.

Titlebuttons specifications must include the name of the pixmap to use in the button box and the function to be invoked when a pointer button is pressed within them:

LeftTitleButton "*bitmapname*" = *function*

or

RightTitleButton "*bitmapname*" = *function*

The *bitmapname* may refer to one of the built-in bitmaps (which are scaled to match **TitleFont**) by using the appropriate colon-prefixed name described above.

Key and pointer button specifications must give the modifiers that must be pressed, over which parts of the screen the pointer must be, and what function is to be invoked. Keys are given as strings containing the appropriate keysym name; buttons are given as the keywords **Button1-Button5**:

```
"FP1"           = modlist : context : function
Button1 = modlist : context : function
```

The *modlist* is any combination of the modifier names **shift**, **control**, and **meta** (which may be abbreviated as **s**, **c**, and **m** respectively) separated by a vertical bar (|). Similarly, the *context* is any combination of **window**, **title**, **icon**, **root**, **frame**, **iconmgr**, their first letters (**iconmgr** abbreviation is **m**), or **all**, separated by a vertical bar. The *function* is any of the **f.** keywords described below. For example, the default startup file contains the following bindings:

```
Button1 =       : root          : f.menu "TwmWindows"
Button1 = m     : window | icon : f.function "move-or-lower"
Button2 = m     : window | icon : f.iconify
Button3 = m     : window | icon : f.function "move-or-raise"
Button1 =       : title         : f.function "move-or-raise"
Button2 =       : title         : f.raiselower
Button1 =       : icon          : f.function "move-or-iconify"
Button2 =       : icon          : f.iconify
Button1 =       : iconmgr       : f.iconify
Button2 =       : iconmgr       : f.iconify
```

A user who wanted to be able to manipulate windows from the keyboard could use the following bindings:

```
"F1"        =       : all           : f.iconify
"F2"        =       : all           : f.raiselower
"F3"        =       : all           : f.warpring "next"
"F4"        =       : all           : f.warpto "xmh"
"F5"        =       : all           : f.warpto "emacs"
"F6"        =       : all           : f.colormap "next"
"F7"        =       : all           : f.colormap "default"
"F20"       =       : all           : f.warptoscreen "next"
"Left"      = m     : all           : f.backiconmgr
"Right" = m | s : all       : f.forwiconmgr
"Up"        = m     : all           : f.upiconmgr
"Down" = m | s  : all       : f.downiconmgr
```

Twm provides many more window manipulation primitives than can be conveniently stored in a titlebar, menu, or set of key bindings. Although a small set of defaults are supplied (unless the **NoDefaults** is specified), most users will want to have their most common operations bound to key and button strokes. To do this, *twm* associates names with each of the primitives and provides *user-defined functions* for building higher level primitives and *menus* for interactively selecting among groups of functions.

User-defined functions contain the name by which they are referenced in calls to **f.function** and a list of other functions to execute. For example:

```
Function "move-or-lower" { f.move f.deltastop f.lower }
Function "move-or-raise"  { f.move f.deltastop f.raise }
Function "move-or-iconify"       { f.move f.deltastop f.iconify }
Function "restore-colormap"      { f.colormap "default" f.lower }
```

The function name must be used in **f.function** exactly as it appears in the function specification.

In the descriptions below, if the function is said to operate on the selected window, but is invoked from a root menu, the cursor will be changed to the Select cursor and the next window to receive a button press will be chosen:

! *string* This is an abbreviation for **f.exec** *string*.

f.autoraise
This function toggles whether or not the selected window is raised whenever entered by the pointer. See the description of the variable **AutoRaise**.

f.backiconmgr
This function warps the pointer to the previous column in the current icon manager, wrapping back to the previous row if necessary.

f.beep This function sounds the keyboard bell.

f.bottomzoom
This function is similar to the **f.fullzoom** function, but resizes the window to fill only the bottom half of the screen.

f.circledown
This function lowers the top-most window that occludes another window.

f.circleup
This function raises the bottom-most window that is occluded by another window.

f.colormap *string*
This function rotates the colormaps (obtained from the WM_COLORMAP_WINDOWS property on the window) that *twm* will display when the pointer is in this window. The argument *string* may have one of the following values: **"next"**, **"prev"**, and **"default"**.

f.deiconify
: This function deiconifies the selected window. If the window is not an icon, this function does nothing.

f.delete This function sends the WM_DELETE_WINDOW message to the selected window if the client application has requested it through the WM_PROTOCOLS window property. The application is supposed to respond to the message by removing the indicated window. If the window has not requested WM_DELETE_WINDOW messages, the keyboard bell will be rung indicating that the user should choose an alternative method.

f.deltastop
: This function allows a user-defined function to be aborted if the pointer has been moved more than *MoveDelta* pixels. See the example definition given for **Function "move-or-raise"** at the beginning of the section.

f.destroy
: This function instructs the X server to close the display connection of the client that created the selected window. This should only be used as a last resort for shutting down runaway clients.

f.downiconmgr
: This function warps the pointer to the next row in the current icon manger, wrapping to the beginning of the next column if necessary.

f.exec *string*
: This function passes the argument *string* to /bin/sh for execution. In multiscreen mode, if *string* starts a new X client without giving a display argument, the client will appear on the screen from which this function was invoked.

f.focus This function toggles the keyboard focus of the server to the selected window, changing the focus rule from pointer-driven if necessary. If the selected window already was focused, this function executes an **f.unfocus**.

f.forcemove
: This function is like **f.move** except that it ignores the **DontMoveOff** variable.

f.forwiconmgr
: This function warps the pointer to the next column in the current icon manager, wrapping to the beginning of the next row if necessary.

f.fullzoom
This function resizes the selected window to the full size of the display or else restores the original size if the window was already zoomed.

f.function *string*
This function executes the user-defined function whose name is specified by the argument *string*.

f.hbzoom
This function is a synonym for **f.bottomzoom**.

f.hideiconmgr
This function unmaps the current icon manager.

f.horizoom
This variable is similar to the **f.zoom** function except that the selected window is resized to the full width of the display.

f.htzoom
This function is a synonym for **f.topzoom**.

f.hzoom This function is a synonym for **f.horizoom**.

f.iconify This function iconifies or deiconifies the selected window or icon, respectively.

f.identify
This function displays a summary of the name and geometry of the selected window. Clicking the pointer or pressing a key in the window will dismiss it.

f.lefticonmgr
This function similar to **f.backiconmgr** except that wrapping does not change rows.

f.leftzoom
This variable is similar to the **f.bottomzoom** function but causes the selected window is only resized to the left half of the display.

f.lower This function lowers the selected window.

f.menu *string*
This function invokes the menu specified by the argument *string*. Cascaded menus may be built by nesting calls to **f.menu**.

f.move This function drags an outline of the selected window (or the window itself if the **OpaqueMove** variable is set) until the invoking pointer button is released. Double clicking within the number of milliseconds given by **ConstrainedMoveTime** warps the pointer to the center of the window and constrains the move to be either horizontal or vertical depending on which grid line is crossed. To abort a move, press another button before releasing the first button.

f.nexticonmgr
This function warps the pointer to the next icon manager containing any windows on the current or any succeeding screen.

f.nop This function does nothing and is typically used with the **DefaultFunction** or **WindowFunction** variables or to introduce blank lines in menus.

f.previconmgr
This function warps the pointer to the previous icon manager containing any windows on the current or preceding screens.

f.quit This function causes *twm* to restore the window's borders and exit. If *twm* is the first client invoked from *xdm*, this will result in a server reset.

f.raise This function raises the selected window.

f.raiselower
This function raises the selected window to the top of the stacking order if it is occluded by any windows, otherwise the window will be lowered.

f.refresh This function causes all windows to be refreshed.

f.resize This function displays an outline of the selected window. Crossing a border (or setting **AutoRelativeResize**) will cause the outline to begin to rubber band until the invoking button is released. To abort a resize, press another button before releasing the first button.

f.restart This function kills and restarts *twm*.

f.righticonmgr
This function is similar to **f.nexticonmgr** except that wrapping does not change rows.

f.rightzoom
This variable is similar to the **f.bottomzoom** function except that the selected window is only resized to the right half of the display.

f.saveyourself
This function sends a WM_SAVEYOURSELF message to the selected window if it has requested the message in its WM_PROTOCOLS window property. Clients that accept this message are supposed to checkpoint all state associated with the

window and update the WM_COMMAND property as specified in the ICCCM. If the selected window has not selected for this message, the keyboard bell will be rung.

f.showiconmgr
This function maps the current icon manager.

f.sorticonmgr
This function sorts the entries in the current icon manager alphabetically. See the variable **SortIconManager**.

f.title This function provides a centered, unselectable item in a menu definition. It should not be used in any other context.

f.topzoom
This variable is similar to the **f.bottomzoom** function except that the selected window is only resized to the top half of the display.

f.unfocus
This function resets the focus back to pointer-driven. This should be used when a focused window is no longer desired.

f.upiconmgr
This function warps the pointer to the previous row in the current icon manager, wrapping to the last row in the same column if necessary.

f.vlzoom This function is a synonym for **f.leftzoom**.

f.vrzoom
This function is a synonym for **f.rightzoom**.

f.warpring *string*
This function warps the pointer to the next or previous window (as indicated by the argument *string*, which may be **"next"** or **"prev"**) specified in the **WindowRing** variable.

f.warpto *string*
This function warps the pointer to the window which has a name or class that matches *string*. If the window is iconified, it will be deiconified if the variable **WarpUnmapped** is set or else ignored.

f.warptoiconmgr *string*
This function warps the pointer to the icon manager entry associated with the window containing the pointer in the icon manager specified by the argument *string*. If *string* is empty (i.e. ""), the current icon manager is chosen.

f.warptoscreen *string*
: This function warps the pointer to the screen specified by the argument *string*. *String* may be a number (e.g. **"0"** or **"1"**), the word **"next"** (indicating the current screen plus 1, skipping over any unmanaged screens), the word **"back"** (indicating the current screen minus 1, skipping over any unmanaged screens), or the word **"prev"** (indicating the last screen visited.

f.winrefresh
: This function is similar to the **f.refresh** function except that only the selected window is refreshed.

f.zoom
: This function is similar to the **f.fullzoom** function, except that the only the height of the selected window is changed.

MENUS

Functions may be grouped and interactively selected using pop-up (when bound to a pointer button) or pull-down (when associated with a titlebutton) menus. Each menu specification contains the name of the menu as it will be referred to by **f.menu**, optional default foreground and background colors, the list of item names and the functions they should invoke, and optional foreground and background colors for individual items:

```
Menu "menuname" [ ("deffore":"defback") ]
{
        string1  [ ("fore1":"backn")]      function1
        string2  [ ("fore2":"backn")]      function2
                 .
                 .
                 .
        stringN  [ ("foreN":"backN")]      functionN
}
```

The *menuname* is case-sensitive. The optional *deffore* and *defback* arguments specify the foreground and background colors used on a color display to highlight menu entries. The *string* portion of each menu entry will be the text which will appear in the menu. The optional *fore* and *back* arguments specify the foreground and background colors of the menu entry when the pointer is not in the entry. These colors will only be used on a color display. The default is to use the colors specified by the **MenuForeground** and **MenuBackground** variables. The *function* portion of the menu entry is one of the functions, including any user-defined functions, or additional menus.

There is a special menu named **TwmWindows** which contains the names of all of the client and *twm*-supplied windows. Selecting an entry will cause the **WindowFunction** to be executed on that window. If **WindowFunction** hasn't been set, the window will be deiconified and raised.

ICONS

Twm supports several different ways of manipulating iconified windows. The common pixmap-and-text style may be laid out by hand or automatically arranged as described by the **IconRegion** variable. In addition, a terse grid of icon names, called an icon manager, provides a more efficient use of screen space as well as the ability to navigate among windows from the keyboard.

Neither client-supplied icon windows nor dynamic setting of the icon pixmap are supported (icon name changes will be updated automatically).

An icon manager is a window that contains names of selected or all windows currently on the display. In addition to the window name, a small button using the default iconify symbol will be displayed to the left of the name when the window is iconified. By default, clicking on an entry in the icon manager performs **f.iconify**. To change the actions taken in the icon manager, use the the **iconmgr** context when specifying button and keyboard bindings.

Moving the pointer into the icon manager also directs keyboard focus to the indicated window (setting the focus explicitly or else sending synthetic events **NoTitleFocus** is set). Using the **f.upiconmgr**, **f.downiconmgr** **f.lefticonmgr**, and **f.righticonmgr** functions, the input focus can be changed between windows directly from the keyboard.

BUGS

Lock and Mod2-5 cannot be specified as modifier contexts. The right fix is to add lock, l, mod1 (for completeness), mod2, mod3, mod4, mod5 to the parse and grammar tables, and add a number as a valid key type (so long as it is 1-5).

The resource manager should have been used instead of all of the window lists.

The **IconRegion** variable should take a list.

Double clicking very fast to get the constrained move function will sometimes cause the window to move, even though the pointer is not moved.

If **IconifyByUnmapping** is on and windows are listed in **IconManagerDontShow** but not in **DontIconifyByUnmapping**, they may be lost if they are iconified and no bindings to **f.menu "TwmWindows"** or **f.warpto** are setup.

FILES

$HOME/.twmrc.<screen number>
$HOME/.twmrc
/usr/lib/X11/twm/system.twmrc

ENVIRONMENT VARIABLES

DISPLAY
: This variable is used to determine which X server to use. It is also set during **f.exec** so that programs come up on the proper screen.

HOME
: This variable is used as the prefix for files that begin with a tilde and for locating the *twm* startup file.

SEE ALSO

X(1), Xserver(1), xdm(1), xrdb(1)

COPYRIGHT

Portions copyright 1988 Evans & Sutherland Computer Corporation; portions copyright 1989 Hewlett-Packard Company and the Massachusetts Institute of Technology, See *X(1)* for a full statement of rights and permissions.

AUTHORS

Tom LaStrange, Solbourne Computer; Jim Fulton, MIT X Consortium; Steve Pitschke, Stardent Computer; Keith Packard, MIT X Consortium; Dave Payne, Apple Computer.

U – V

NAME

ul - do underlining

SYNOPSIS

ul [−i] [−t *terminal*] [*name* ...]

DESCRIPTION

ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. The −t option overrides the terminal kind specified in the environment. The file */usr/lib/terminfo* is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

The −i option causes *ul* to indicate underlining onto by a separate line containing appropriate dashes '−'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

ul also handles bold printing, alternate character sets, etc. when the sequences are defined in the terminal's */usr/lib/terminfo* entry.

SEE ALSO

man(1), nroff(1)

AUTHOR

Mark Horton wrote *ul*. The −i option was originally a option of the editor *ex*(1), then an *iul* command.

BUGS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in the user's **.profile** (see *profile*(4)) and invoked at login to automatically set the user's permissions on files or directories created.

SEE ALSO

chmod(1), sh(1).
chmod(2), creat(2), umask(2), profile(4) in the *Programmer's Reference Manual*.

NAME

uname – identify the current IRIX system

SYNOPSIS

uname [**–snrvma**]

uname [–S system-name]

DESCRIPTION

uname prints information that identifies the current the IRIX system to standard output. The options cause selected information returned by *uname*(2) to be printed:

—s	Print the system name (the default).
—n	Print the nodename (the nodename is the name by which the system is known to a communications network).
—r	Print the operating system release. This string has one of the following forms: *m.n* or *m.n.a* where *m* is the major release number, *n* is the minor release number and *a* is the (optional) maintenance level of the release; e.g. **3.2** or **3.2.1**.
—v	Print the operating system version. This is the date and time that the operating system was generated, and has the form: *mmddhhmm*.
—m	Print the machine hardware name. This is the type of CPU board that the system is running on, e.g. **IP6.**
—a	Print all the above information.

The system name and the nodename may be changed by specifying a system name argument to the –S option. The system name argument is restricted to 8 characters. Only the super-user is allowed this capability.

SEE ALSO

uname(2) in the *Programmer's Reference Manual.*

NAME

unget — undo a previous get of an SCCS file

SYNOPSIS

unget [−rSID] [−s] [−n] files

DESCRIPTION

unget undoes the effect of a **get −e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

—r*SID*	Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *gets* for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.
—s	Suppresses the printout, on the standard output, of the intended delta's *SID*.
—n	Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), sact(1).
help(1) in the *User's Reference Manual*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

unifdef – strip or reduce ifdefs in C code

SYNOPSIS

unifdef [**–D***name*] [**–D***name=string*] [**–U***name*] [**–o***output*] [**-z**] [*input...*]

DESCRIPTION

Unifdef reads C source files and prints all input lines except those excluded by **#ifdef** constructs which refer to specified identifiers.

The **–D** option defines *name* as a macro with the value 1, causing *unifdef* to simplify

```
#ifdef name
Included text
#else
Excluded text
#endif
```

to just

```
Included text
```

and contrariwise for **#ifndef.**

The **–U** option works like **–D** except that *name* is undefined, so the **#ifdef** above would simplify to

```
Excluded text
```

-D*name=string* causes *unifdef* to replace occurrences of *name* with *string* when evaluating **#if** and **#elif** expressions. For example, **unifdef –DBSD=43** would rewrite

```
#if BSD == 43
4.3BSD code
#elif BSD == 42
4.2BSD code
#endif
```

as

```
4.3BSD code
```

All **#ifdef** constructs which refer to names not defined with **–D** or undefined with **–U** are passed unchanged to output. *Unifdef* simplifies **#if** expressions which mix defined or undefined names with unknown names to express operations on just the unknown names. Thus **unifdef –Dsgi** would rewrite

```
#if defined sgi && !defined KERNEL
```

as

```
        #ifndef KERNEL
```

and would preserve associated **#else** and **#endif** sections. *Unifdef* simplifies **#elif** chains as if they consisted of **#if** constructs nested within **#else** sections. For example, **unifdef –DBSD=43** would rewrite

```
        #if SYSV
        System V code
        #elif BSD == 42
        4.2BSD code
        #elif BSD == 43
        4.3BSD code
        #elif XENIX
        Xenix code
        #endif
```

as

```
        #if SYSV
        System V code
        #else
        4.3BSD code
        #if XENIX
        Xenix code
        #endif
        #endif
```

Unifdef rewrites **#else** and **#endif** lines to conform to ANSI C, by enclosing any tokens after these keywords with comment delimiters. For example,

```
        #ifdef DEBUG
        Debugging code
        #endif DEBUG
```

would be transcribed as

```
        #ifdef DEBUG
        Debugging code
        #endif /* DEBUG */
```

Unifdef writes to standard output by default. The **–o** option may be used in lieu of shell redirection to specify an output file. More important, the file specified by *output* may be identical to one of the *input* files, or to standard input if there are no arguments. In this case, *unifdef* will safely overwrite the input file.

The **−z** option causes *unifdef* to simplify or eliminate sections controlled by #if constructs that test constant integer expressions. By default, constructs such as

```
#if 0
Excluded text
#endif
```

are left intact.

AUTHOR

Brendan Eich, 03/16/89

SEE ALSO

cc(1), cpp(1).

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**–udc** [+n] [–n]] [input [output]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **—u** flag is used, just the lines that are not repeated in the original file are output. The **–d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **–u** and **–d** mode outputs.

The **–c** option supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
          * 2.540000e+00
          / 3.937008e–01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
          * 1.020689e+00
          / 9.797299e–01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as g,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME
uptime – show how long system has been up

SYNOPSIS
uptime

DESCRIPTION
Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

FILES
/unix	system name list

SEE ALSO
w(1)

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file
uulog [options] –ssystem
uulog [options] system
uulog [options] –fsystem
uuname [–l] [–c]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names that *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to foward information (see WARNINGS below for restrictions).

The shell metacharacters ?, * and [...] appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ˜*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ˜*/destination* where *destination* is appended to **/usr/spool/uucppublic**; (NOTE: This destination will be treated as a file name unless more than one file is being transfered by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ˜/dan/ as the destination will make the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory).

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

−c	Do not copy local file to the spool directory for transfer to the remote machine (default).
−C	Force the copy of local files to the spool directory for transfer.
−d	Make all necessary directories for the file copy (default).
−f	Do not make intermediate directories for the file copy.
−g*grade*	*Grade* is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.
−j	Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.
−m	Send mail to the requester when the copy is completed.
−n*user*	Notify *user* on the remote system that a file was sent.
−r	Do not start the file transfer, just queue the job.
−s*file*	Report status of the transfer to *file*. Note that the *file* must be a full path name.
−x*debug_level*	Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiles with -DSMALL.)

uulog

uulog queries a log file of *uucp* or *uuxqt* transactions in a file **/usr/spool/uucp/.Log/uucico/***system*, or **/usr/spool/uucp/.Log/uuxqt/***system*.

The options cause *uulog* to print logging information:

−s*sys*	Print information about file transfer work involving system *sys*.

−f*system*	Does a "tail −f" of the file transfer log for *system*. (You must hit BREAK to exit this function.) Other options used in conjunction with the above:
−x	Look in the *uuxqt* log file for the given system.
−*number*	Indicates that a "tail" command of *number* lines should be executed.

uuname

uuname lists the names of systems known to *uucp*. The −c option returns the names of systems known to *cu*. (The two lists are the same, unless your machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.) The −l option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (/usr/spool/uucppublic)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uustat(1C), uucico(1M), uux(1C), uuxqt(1M).
chmod(2) in the *Programmer's Reference Manual.*

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin /usr/spool/uucppublic (equivalent to ~/).

All files received by *uucp* will be owned by *uucp*.
The −m option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the −m option.

The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent by *uucp*. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uuencode, uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [source] remotedest | **mail** sys1!sys2!..!decode
uudecode [file]

DESCRIPTION

Uuencode and *uudecode* are used to send a binary file via uucp (or other) mail. This combination can be used over indirect mail links even when *uusend*(1C) is not available.

Uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user ''decode'' should be filtered through the *uudecode* program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using *sendmail* or by making *rmail* be a link to *Mail* instead of *mail*. In each case, an alias must be created in a master file to get the automatic invocation of *uudecode*.

If these facilities are not available, the file can be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

SEE ALSO

uucp(1C), uux(1C), mail(1), uuencode(4)

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often *uucp*) must have write permission on the specified file.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [–a]
uustat [–m]
uustat [–p]
uustat [–q]
uustat [–k*jobid*]
uustat [–r*jobid*]
uustat [–s*system*] [–u*user*]

DESCRIPTION

uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

–a	Output all jobs in queue.
–m	Report the status of accessibility of all machines.
–p	Execute a "ps –flp" for all the process-ids that are in the lock files.
–q	List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of C or X files, it is the age in days of the oldest C./X. file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the –q option:

```
eagle     3C   04/07-11:07   NO DEVICES AVAILABLE
mh3bs3    2C   07/07-10:42   SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

–k*jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

−r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with *uustat*:

−s*sys*	Report the status of all *uucp* requests for remote system *sys*.
−u*user*	Report the status of all *uucp* requests issued by *user*.

Output for both the −s and −**u** options has the following format:

```
eaglen0000  4/07-11:01:03        (POLL)
eagleN1bd7  4/07-11:07           Seagledan522 /usr/dan/A
eagleC1bd8  4/07-11:07           Seagledan59 D.3b2al2ce4924
                  4/07-11:07     Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/*   spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [options] source-files destination
uupick [–s system]

DESCRIPTION

uuto sends *source-files* to *destination*. *uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *User* is the login name of someone on the specified system.

Two *options* are available:

–p	Copy the source file into the spool directory before transmission.
–m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system* , where PUBDIR is a public directory defined in the *uucp* source. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] **?**

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** except moving all the files sent from *system*.

p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*****	Print a command summary.

Uupick invoked with the -s*system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR /usr/spool/uucppublic public directory

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C).
uucleanup(1M) in the *System Administrator's Reference Manual*.

WARNINGS

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

NOTE: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permitting only the receipt of mail (see *mail*(1)). (Remote execution permissions are defined in /usr/**lib/uucp/Permissions.**)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the *file1* and *file2* files from the "usg" and "pwba" machines, execute a *diff*(1) command and put the results in *file.diff* in the local PUBDIR/dan/ directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets /usr/file from system "b" and sends it to system "a", performs a *cut* command on that file and sends the result of the *cut* command to system "c".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **−n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

−	The standard input to *uux* is made the standard input to the *command-string*.
−a*name*	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
−b	Return whatever standard input was provided to the *uux* command if the exit status is non-zero.
−c	Do not copy local file to the spool directory for transfer to the remote machine (default).
−C	Force the copy of local files to the spool directory for transfer.
−g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
−j	Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.
−n	Do not notify the user if the command fails.
−p	Same as −: The standard input to *uux* is made the standard input to the *command-string*.
−r	Do not start the file transfer, just queue the job.
−s*file*	Report status of the transfer in *file*.
−x*debug_level*	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
−z	Send success notification to the user.

FILES

/usr/lib/uucp/spool	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work. (If *diff* is a permitted command.)

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using *uux*. However, if the requestor is root, and the directory is not searchable by "other", the request will fail.

NAME

uwm - a window manager for X

SYNTAX

uwm [-display *display*] [-f *filename*]

DESCRIPTION

The *uwm* program is a window manager for X.

When *uwm* is invoked, it searches a predefined search path to locate any *uwm* startup files. If no startup files exist, *uwm* initializes its built-in default file.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the *uwm* search path are:

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

To use only the settings defined in a single startup file, include the variables **resetbindings**, **resetmenus**, **resetvariables** at the top of that specific startup file.

OPTIONS

-f *filename*
: Names an alternate file as a *uwm* startup file.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings**, **resetmenus**, and **resetvariables** head the list.

autoselect/noautoselect
: places the menu cursor in the first menu item. If unspecified, the menu cursor is placed in the menu header when the menu is displayed.

background=*color*
: specifies the default background color for popup sizing windows, menus, and icons. The default is to use the WhitePixel for the current screen.

bordercolor=*color*
: specifies the default border color for popup sizing windows, menus, and icons. The default is to use the BlackPixel for the current screen.

borderwidth=*pixels*
specifies the default width in pixels for borders around popup sizing windows, menus, and icons. The default is 2.

delta=*pixels* indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)

foreground=*color*
specifies the default foreground color for popup sizing windows, menus, and icons. The default is to use the BlackPixel for the current screen.

freeze/nofreeze locks all other client applications out of the server during certain window manager tasks, such as move and resize.

grid/nogrid displays a finely-ruled grid to help you position an icon or window during resize or move operations.

hiconpad=*pixels* indicates the number of pixels to pad an icon horizontally. The default is five pixels.

hmenupad=*pixels*
indicates the amount of space in pixels that each menu item is padded to the left and to the right of the text.

borderwidth=*pixels*
indicates the width in pixels of the border surrounding icons.

iconfont=*fontname*
names the font that is displayed within icons. Font names for a given server can be obtained using *xlsfonts(1)*.

maxcolors=*number*
limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, *uwm* assumes no limit to the number of colors it can take from the color map. **maxcolors** counts colors as they are included in the file.

mborderwidth=*pixels*
indicates the width in pixels of the border surrounding menus.

normali/nonormali
places icons created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormali** the icon is placed exactly where the

cursor leaves it.

normalw/nonormalw
places window created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*number* moves a window *number* pixels or 1/*number* times the size of the window, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup, f.pushdown, f.pushright**, or **f.pushleft.**

pushabsolute/pushrelative
pushabsolute indicates that the number entered with push is equivalent to pixels. When an f.push (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

pushrelative indicates that the number entered with the push variable represents a relative number. When an f.push function is called, the window is invisibly divided into the number of parts you entered with the **push** variable, and the window is moved one part.

resetbindings, resetmenus, and **resetvariables**
resets all previous function bindings, menus, and variable entries, specified in any startup file in the *uwm* search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname*
identifies the font of the indicator that displays dimensions in the corner of the window as you resize windows. See *xlsfonts(1)* for obtaining font names.

resizerelative/noresizerelative
indicates whether or not resize operations should be done relative to moving edge or edges. By default, the dynamic rectangle uses the actual pointer location to define the new size.

reverse/noreverse
defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*pixels*	indicates the number of pixels to pad an icon vertically. Default is five pixels.
vmenupad=*pixels*	indicates the amount of space in pixels that the menu is padded above and below the text.
volume=*number*	increases or decreases the base level volume set by the *xset(1)* command. Enter an integer from 0 to 7, 7 being the loudest.
zap/nozap	causes ghost lines to follow the window or icon from its previous default location to its new location during a move or resize operation.

BINDING SYNTAX

function=[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "

Function and mouse events are required input. Menu name is required with the *f.menu* function definition only.

Function

f.beep	emits a beep from the keyboard. Loudness is determined by the volume variable.
f.circledown	causes the top window that is obscuring another window to drop to the bottom of the stack of windows.
f.circleup	exposes the lowest window that is obscured by other windows.
f.continue	releases the window server display action after you stop action with the **f.pause** function.
f.focus	directs all keyboard input to the selected window. To reset the focus to all windows, invoke *f.focus* from the root window.
f.iconify	when implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.
f.kill	kills the client that created a window.
f.lower	lowers a window that is obstructing a window below it.
f.menu	invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses. f.menu=[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "

f.move	moves a window or icon to a new location, which becomes the default location.
f.moveopaque	moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.
f.newiconify	allows you to create a window or icon and then position the window or icon in a new default location on the screen.
f.pause	temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.
f.pushdown	moves a window down. The distance of the push is determined by the push variables.
f.pushleft	moves a window to the left. The distance of the push is determined by the push variables.
f.pushright	moves a window to the right. The distance of the push is determined by the push variables.
f.pushup	moves a window up. The distance of the push is determined by the push variables.
f.raise	raises a window that is being obstructed by a window above it.
f.refresh	results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows will not refresh correctly if the exposure events are not handled properly.
f.resize	resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.
f.restart	causes the window manager application to restart, retracing the *uwm* search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one, two, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager may use pointer button and control key combinations. If the window manager has bound these combinations for its own use, the client application will never see the requested pointer input.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null).

The root context refers to the root, or background window, A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lower case and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down	function occurs when the specified button is pressed down.
up	function occurs when the specified button is released.
delta	indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

```
menu = " menu name " {
"item name" : "action"
      .
      .
      .
}
```

Enter the menu name exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the f.menu function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions
: Any function previously described. E.g., **f.move** or **f.iconify**.

Shell commands
: Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings
: Text strings are placed in the window server's cut buffer.

  Strings starting with an up arrow (^) will have a new line character appended to the string after the up arrow (^) has been stripped from it.

  Strings starting with a bar character (|) will be copied as is after the bar character (|) has been stripped.

Color Menus

Use the following syntax to add color to menus:

```
menu = "menu name" (color1:color2:color3:color4) {
"item name"  : (color5 :color6)  : " action "
     .
     .
     .
}
```

color1	Foreground color of the header.
color2	Background color of the header.
color3	Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.
color4	Background color of the highlighter.
color5	Foreground color for the individual menu item.
color6	Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of color map entries, either before or during an invocation of *uwm*.

2) If you specify a foreground or background color that does not exist in the RGB color database of the server (see */usr/lib/X11/rgb.txt* for a sample) both the foreground and background colors default to the root window colors.

3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.

4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.

5) If you specify no colors in the startup file.

Customizing Icon Names

Icon names may be editted by placing the pointer inside the icon and typing. The Backspace, Rubout and Delete keys may used to remove a character from the end of a line and Control-U may be used to delete the whole name.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
freeze
grid
hiconpad=5
hmenupad=6
iconfont=oldeng
menufont=timrom12b
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
```

```
#
# FUNCTION   KEYS  CONTEXT  BUTTON    MENU(if any)
# ========   ====  =======  ======    ===========
f.menu =     meta  :        :left down   :"WINDOW OPS"
f.menu =     meta  :        :middle down :"EXTENDED WINDOW OPS"
f.move =     meta  :wli     :right down
f.circleup = meta  :root    :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
"(De)Iconify":   f.iconify
Move:            f.move
Resize:          f.resize
Lower:           f.lower
Raise:           f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:                 !"xterm &"
Iconify at New Position:    f.lowericonify
Focus Keyboard on Window:      f.focus
Freeze All Windows:            f.pause
Unfreeze All Windows:          f.continue
Circulate Windows Up:          f.circleup
Circulate Windows Down:        f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

Some versions of lex have a hard-wired input buffer limit of 200 characters with no checking for overflow. A .uwmrc file containing lines longer than this limit will cause unpredictable behavior when used with a version of uwm built on such a system.

FILES

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

SEE ALSO

X(1), Xserver(1), xset(1), xlsfonts(1)

COPYRIGHT

AUTHOR

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science.

NAME

vadmin – interactive system administration tool

SYNOPSIS

vadmin [**–x** xpos] [**–y** ypos] [**–w** width] [**–h** height] [**–t** title] [**–d** helpfile] [**–s**] [tools-directory]

DESCRIPTION

The *vadmin* command is used to display a set of executable files contained in *tools-directory*. *vadmin* scans through the *tools-directory* and displays those executable files as a collection of selectable icons is a window. The default *tools-directory* used is */usr/lib/vadmin*.

Command line options are available to specify the window size, title and position as well as the help textfile (*vhelp*(1)). The command line options include:

–x *xpos*	right justify the window at *xpos*. The –x option, when used in conjunction with the –y option, define the position of the window's lower-left corner to the screen pixel location *xpos,ypos*.
–y *ypos*	aligns the bottom window edge at *ypos*. The –y option, when used in conjunction with the –x option, define the position of the window's lower-left corner to the screen pixel location *xpos,ypos*.
–w *width*	sets the window's width to *width*.
–h *height*	sets the window's height to *height*.
–t *title*	sets the window title to *title*.
–s	omits the "Invoke tools as" option. *vadmin* will bypass checking for uid and execute files as the current uid.
–d *helpfile*	uses the textfile *helpfile* as the text to be displayed when the help option is selected from the window. The *helpfile* is displayed by using *vhelp*(1). The default helpfile is */usr/lib/vadmin/vadmin.hlp*.

FILES

/usr/lib/vadmin
/usr/lib/vadmin/filetypes

SEE ALSO

vhelp(1)

NAME

val – validate SCCS file

SYNOPSIS

val –

val [–s] [–rSID] [–mname] [–ytype] files

DESCRIPTION

val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a –, and named files.

val has a special argument, –, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

–s	The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.
–r*SID*	The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.
–mname	The argument value *name* is compared with the s-1SCCS %M% keyword in *file*.
–ytype	The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **−y** mismatch;
bit 7 = %M%, **−m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

val can process up to 50 files on a single command line. Any number above 50 will produce a core dump.

NAME

vc – version control

SYNOPSIS

vc [–a] [–t] [–cchar] [–s] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the –t keyletter (see below). The default control character is colon (:), except as modified by the –c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The –a keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

–a	Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.
–t	All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.

−cchar	Specifies a control character to be used in place of :.
−s	Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

```
:if condition
        .
        .
        .
:end
```

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.
The syntax of a condition is:

```
<cond>          ::= [ "not" ] <or>
<or>            ::= <and> | <and> "|" <or>
<and>           ::= <exp> | <exp> "&" <and>
<exp>           ::= "(" <or> ")" | <value> <op> <value>
<op>            ::= "=" | "!=" | "<" | ">"
<value>         ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
        = != > <      all of equal precedence
        &
        |
```

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.

::text
: Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the –a keyletter.

:on

:off
: Turn on or off keyword replacement on all lines.

:ctl char
: Change the control character to char.

:msg message
: Prints the given message on the diagnostic output.

:err message
: Prints the given message followed by:

    ERROR: err statement on line ... (915)

  on the diagnostic output. *vc* halts execution, and returns an exit code of 1.

SEE ALSO
: ed(1), help(1) in the *User's Reference Manual.*

DIAGNOSTICS
: Use *help*(1) for explanations.

EXIT CODES
: 0 – normal
  1 – any error

NAME

vi, view, vedit – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**–t** tag] [**–r** file] [–L] [–wn] [–R] [–x] [–C] [**–c** command] file ...
view [**–t** tag] [**–r** file] [–L] [–wn] [–R] [–x] [–C] [**–c** command] file ...
vedit [**–t** tag] [**–r** file] [–L] [–wn] [–R] [–x] [–C] [**–c** command] file ...

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all *ex*(1) line editor commands are described on the *ex*(1) manual page.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

Invocation Options

The following invocation options are interpreted by *vi* (previously documented options are discussed in the **NOTES** section at the end of this manual page):

–t *tag*	Edit the file containing the *tag* and position the editor at its definition.
–r *file*	Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)
–L	List the name of all files saved as the result of an editor or system crash.
–w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
–R	**Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.
–x	Encryption option; when used, *vi* simulates the X command of *ex*(1) and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option.

See *crypt*(1). Also, see the **WARNING** section at the end of this manual page.

–C
: Encryption option; same as the **–x** option, except that *vi* simulates the C command of *ex*(1). The C command is like the **X** command of *ex*(1), except that all text read in is assumed to have been encrypted.

–c *command*
: Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. It is the same as *vi* except that the **report** flag is set to 1, the **showmode** and **novice** flags are set, and **magic** is turned off. These defaults make it easier to learn how to use *vi*.

vi Modes

Command
: Normal and initial mode. Other modes return to command mode upon completion. **ESC** (escape) is used to cancel a partial command.

Input
: Entered by setting any of the following options: **a A i I o O c C s S R .** Arbitrary text may then be entered. Input mode is normally terminated with **ESC** character, or, abnormally, with an interrupt.

Last line
: Reading input for **: / ?** or **!**; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

In the descriptions, CR stands for carriage return and ESC stands for the escape key.

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text***ESC**	insert *text*
cw*new***ESC**	change word to *new*
ea*s***ESC**	pluralize word (end of word; append s; escape from input state)
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete 3 lines
u	undo previous change
ZZ	exit *vi*, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*cmd***CR**	any *ex* or *ed* command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
DEL	(delete or rubout) interrupts

File manipulation

ZZ	if file modified, write and exit; otherwise, exit
:wCR	write back changes
:w!CR	forced write, if permission originally not valid
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file

:e ! #CR	edit alternate file, discard changes
:w *name*CR	write file *name*
:w ! *name*CR	overwrite file *name*
:shCR	run shell, then return
:!*cmd*CR	run *cmd*, then return
:nCR	edit next file in arglist
:n *args*CR	specify new arglist
^G	show current file and line
:tag *tag*CR	position cursor to *tag* (see *ctags*(1)), save position
:popCR	return to previous tag's position

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
*n*G	go to the beginning of the specified line (end default), where *n* is a line number
/*pat*	next line matching *pat*
?*pat*	previous line matching *pat*
n	repeat last / or ? command
N	reverse last / or ? command
/*pat*/+*n*	nth line after *pat*
?*pat*?-*n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }
^]	:tag command using word after the cursor as the tag
^T	return to previous tag's position (:pop command)

Adjusting the screen

^L	clear and redraw window
^R	clear and redraw window if ^L is → key
zCR	redraw screen with current line at top of window
z–CR	redraw screen with current line at bottom of window
z.CR	redraw screen with current line at center of window
/*pat*/z–CR	move *pat* line to bottom of window

z*n*.CR	use *n*-line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``	move cursor to previous context
´´	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lower-case letter *x*
`*x*	move cursor to mark *x*
´*x*	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

^	first non white-space character
0	beginning of line
$	end of line
l or →	forward
h or ←	backward
^H	same as ← (backspace)
space	same as → (space bar)
f*x*	find next *x*
F*x*	find previous **x**
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last **f F t** or **T**
,	repeat inverse of last **f F t** or **T**
n|	move to column *n*
%	find matching ({) or }

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
}	to next paragraph

(	back a sentence
{	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Corrections during insert

^H	erase last character (backspace)
^W	erase last word
erase	your erase character, same as **^H** (backspace)
kill	your kill character, erase this line of input
\	quotes your erase and kill characters
ESC	ends insertion, back to command mode
DEL	interrupt, terminates insert mode
^D	backtab one character; reset left margin of *autoindent*
^^D	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*
0^D	backtab to beginning of line; reset left margin of *autoindent*
^T	insert *shiftwidth* spaces.
^V	quote non-printable character

Insert and replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since w moves over a word, dw deletes the word that would be moved over. Double the operator, e.g., dd to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift

>	right shift
!	filter through command

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	delete characters before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters **a** - **z**), the text in that buffer is put instead.

3yy	yank 3 lines
3yl	yank 3 characters
p	put back text after cursor
P	put back text before cursor
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d*p	retrieve *d*'th last delete

AUTHOR

vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/tmp	default directory where temporary work files are placed; it can be changed using the directory option (see the *ex(1)* set command)
/usr/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.COREterm/?/*	subset of compiled terminal description database

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see *intro*(1)). A **–r** option that is not followed with an option-argument has been replaced by **–L** and *+command* has been replaced by **–c** *command*.

SEE ALSO

ctags(1), ed(1), edit(1), ex(1).
User's Guide.
curses/terminfo chapter of the *Programmer's Guide*.

WARNINGS

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

Tampering with entries in */usr/lib/.COREterm/?/** or */usr/lib/terminfo/?/** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

BUGS

In insert mode, software tabs using **^T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

vmsprep – VMS tape preparation aid

SYNOPSIS

vmsprep [–] [name ...]

DESCRIPTION

Vmsprep traverses hierarchies of files and prepares them for transportation to VMS. Since ANSI stardard tapes (the VMS standard) do not allow hierarchy, this program provides a method of flattening the hierarchy onto a tape in such a way that it can be unpacked on VMS to recreate the same tree structure.

For reasons best not described here, *vmsprep* will attempt to exclude all RCS and SCCS archives by ignoring all files or directories named 'RCS' or 'SCCS', or files starting with 's.' or ending in ',v'.

The output of *vmsprep* is a pair of files vmsprep.namelist and UNPACK.COM. vmsprep.namelist is a list of files to be placed on the tape in the format required by *ansitape*. If the first argument is '–' instead of a file or directory name, vmsprep will instead send the namelist to standard output, and place UNPACK.COM in /tmp to avoid attempting to write in the current directory. All of the files except UNPACK.COM will be placed on the tape under cryptic names. UNPACK.COM is a VMS command script which will recreate all of the necessary directories and then move the cryptically named files to their proper place.

A typical sequence would be:

```
vmsprep – tree1 tree2 file | ansitape cln trees –
```

Then on the VMS machine

```
mount MFA0: trees
copy MFA0:*.*.* *
@UNPACK
```

FILES

vmsprep.namelist
UNPACK.COM

DIAGNOSTICS

A warning is reported if a file or directory name contains a character not permitted in VMS names. The offending character is replaced by 'Z' and *vmsprep* continues.

SEE ALSO

ansitape(1)

BUGS

Extra periods in file names may not be dealt with optimally.

All files and directories to be moved must be descendants of the current working directory. Absolute path names and paths containing ".." will produce unpredictable results.

Since vmsprep uses find(1) internally, it does not follow symbolic links.

The exclusion of RCS and SCCS files should be controlled by a command line flag.

Assumes VMS v4.0 or greater for long file names.

ORIGIN

4.3BSD

W – Z

NAME

w – who is on and what they are doing

SYNOPSIS

w [**–fhls**] [user]

DESCRIPTION

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the host from which the user is logged in, the time the user logged on, the time since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

The **–h** flag suppresses the heading. The **–s** flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. **–l** gives the long output, which is the default. The **–f** option suppresses the ''from'' field.

If a *user* name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem

SEE ALSO

who(1), ps(1)

AUTHOR

Mark Horton

BUGS

The notion of the ''current process'' is muddy. The current algorithm is ''the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal''. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, *w* prints ''–''.)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is ''charged'' with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

NAME

wait – await completion of process

SYNOPSIS

wait [*n*]

DESCRIPTION

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wall – write to all users

SYNOPSIS

/etc/wall [-g group] [msg_file]

DESCRIPTION

wall reads *msg_file* (or its standard input if none specified) until an end-of-file. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

If the *-g* option is used then only those users in the specified group are sent the message. The list of logged-on users is derived from the file **/etc/utmp**.

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked (see *mesg*(1)).

FILES

/dev/*
/etc/utmp

SEE ALSO

mesg(1), write(1), who(1).

DIAGNOSTICS

''Cannot send to ...'' when the open on a user's tty file fails.

NAME

wc – word count

SYNOPSIS

wc [**–lwc**] [names]

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **–lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what – identify SCCS files

SYNOPSIS

what [—s] files

DESCRIPTION

what searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information ";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
          identification information
f.o:
          identification information
a.out:
          identification information
```

what is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

—s Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use *help*(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

whatis – describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

whatis looks up a given command and gives the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'whatis ed' and then you should do 'man 1 ed' to get the manual.

whatis is actually just the **–f** option to the *man*(1) command.

FILES

/usr/catman/whatis	Data base

SEE ALSO

apropos(1), man(1).

NAME

whereis - locate source, binary, and or manual for program

SYNOPSIS

whereis [**–sbm**] [**–u**] [**–SBM** dir ... **–f**] name ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of source code control are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places. If any of the **–b**, **-s** or **–m** flags are given then *whereis* searches only for binaries, sources or manual sections respectively (or any two thereof). The **–u** flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory which have no documentation.

Finally, the **–B** , **–M** , and **–S** flags may be used to change or otherwise limit the places where *whereis* searches. The **–f** file flags is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/u_man/man1 with source in /usr/src/cmd:

```
cd /usr/bin
whereis –u –M /usr/man/u_man/man1 –S /usr/src/cmd –f *
```

FILES

/bin, /etc, /lib
/usr/{bin, sbin, bsd, etc, games, demos, lib, lbin}
/usr/local/{bin, etc, lib}
/usr/bin/X11
/usr/man/u_man/{man1, man2, man3, man4, man5, man6}
/usr/man/a_man/{man1, man7}
/usr/src/cmd

BUGS

Since the program uses *chdir*(2) to run faster, pathnames given with the **–M –S** and **–B** must be full; i.e. they must begin with a "/".

NAME

which – locate a program file including aliases and path (*csh* only)

SYNOPSIS

which [–a] [–f] [*name...]*

DESCRIPTION

Which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Aliases are taken from the user's .cshrc file. The current value of path is used. The –a option reports all instances rather than just the first one. With the **–f** (fast) option, *which* ignores the .cshrc file.

FILES

~/.cshrc	source of aliases

DIAGNOSTICS

A diagnostic is given for names that are aliased to one or more words, or if an executable file with the argument name was not found in the path.

BUGS

Must be executed by a csh, since only csh's know about aliases.

NAME

who – who is on the system

SYNOPSIS

who [**–uTlHqpdbrtas**] [**–n** *x*] [file] [file]

who am i

who am I

DESCRIPTION

who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the **/etc/utmp** file at login time to obtain its information. If *file* is given, that file (which must be in *utmp*(4) format) is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

who with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

    name [state] line time [idle] [pid] [comment] [exit]

The *name*, *line*, and *time* information is produced by all options except **–q**; the *state* information is produced only by **–T**; the *idle* and *pid* information is produced only by **–u** and **–l**; and the *comment* and *exit* information is produced only by **–a**. The information produced for **–p**, **–d**, and **–r** is explained during the discussion of each option, below.

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

–u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (**.**) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

–T This option is the same as the **–s** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a – appears if it is not. **root** can write to all lines having a + or a – in the *state* field. If a bad line is encountered, a ? is printed.

–l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

–H This option will print column headings above the regular output.

–q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

–p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

–d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

–b This option indicates the time and date of the last reboot.

–r This option indicates the current *run-level* of the *init* process. In addition, it produces the process termination status, process id, and process exit status (see *utmp*(4)) under the *idle*, *pid*, and *comment* headings, respectively.

–t This option indicates the last change to the system clock (via the *date*(1) command) by root. See *su*(1).

–a This option processes **/etc/utmp** or the named *file* with all options turned on.

–s This option is the default and lists only the *name*, *line*, and *time* fields.

–n *x* This option takes a numeric argument, *x*, which specifies the number of users to display per line. *x* must be at least **1**. The **–n** option must be used with **–q**.

Note to the super-user: after a shutdown to the single-user state, *who* returns a prompt; the reason is that since **/etc/utmp** is updated at login time and there is no login in single-user state, *who* cannot report accurately on this state. *who am i*, however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), su(1M).
init(1M) in the *System Administrator's Reference Manual.*
wait(2), inittab(4), utmp(4) in the *Programmer's Reference Manual.*

NAME

whoami – print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd	Name data base

SEE ALSO

who (1)

NAME

winterm – utility to launch applications that require a terminal emulator.

SYNOPSIS

winterm [**–H**|**–f** font|**–t** title|**–p** x,y|**–s** cols,lines|**–c** command]

DESCRIPTION

winterm is a shell script that presents an abstract command line syntax for the user's own terminal emulator. Terminal emulators supported include *wsh*, *psterm*, and *xterm*. The user can preset their preferred termulator (with preferred options) by setting the environment variable *$WINTERM*. If *WINTERM* is unset, winterm provides *wsh* as a default.

–H	holds the winterm open.
–f font	sets the font used by the winterm to *font*.
–t title	sets the title used by the winterm to *title*
–p x,y	sets the position of lower left corner of the winterm to *x,y*.
–s cols,lines	sets the size of the winterm to *cols,lines*.
–c command	feeds the rest of the line as the command to execute. Must be the last flag set when winterm is invoked.

NOTES

The default *WINTERM* is: `WINTERM='wsh -fScreen11 -C54, 96,3,2,0,50'`.
The **–H** option is not supported by *xterm*.
The **–f** option is not supported by *psterm*.

SEE ALSO

Programming the IRIS WorkSpace

NAME

WorkSpace – graphical interface to file system

SYNOPSIS

workspace [–w] [**–t** [name]]

DESCRIPTION

WorkSpace provides a graphical, interactive interface to the IRIX file system. This interface is provided via two kinds of views. When invoked with no arguments, WorkSpace opens a window displaying a portion of the IRIX file tree, which can be pruned and expanded on a per user basis. The second type of view, instantiated by opening a directory icon, provides an constantly up-to-date representation of that
IRIX directory.

WorkSpace accepts the following options.

–w Open only the WorkSpace view (Note that otherwise WorkSpace will start up with whatever views were open the last time it was used).

–t [name]
Allow the user to create a sample WorkSpace view, and install it in /usr/lib/workspace. If a *name* is provided after this option, the file will be installed in the form *name*.wsrc. Otherwise, the file will be called *$USER*.wsrc. Each time a new template is added, or one is altered, all users on the system will automatically load it the next time they start their WorkSpace. Note that the template file will have to be installed manually if this option is run by a user without privilege to write into */usr/lib/workspace*.

Note that only one WorkSpace process can be run for each user. If the process is invoked again while it is already running, the WorkSpace window will be either opened or popped to the top.

SEE ALSO

dirview(1G)

Programming the IRIS WorkSpace

Author

Betsy Zeller

NAME

write – write to another user

SYNOPSIS

write user [line]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

> **Message from** *yourname* **(tty??)** [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

> *user* is logged on more than one place.
> You are connected to "*terminal*".
> Other locations are:
> *terminal*

Permission to write may be denied or granted by use of the *mesg*(1) command. Writing to others is normally allowed by default. Certain commands, such as *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute !

SEE ALSO

mail(1), mesg(1), pr(1), sh(1), who(1).

DIAGNOSTICS

"*user is not logged on*" if the person you are trying to *write* to is not logged on.

"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).

"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.

"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

wsh – creates and specifies a window shell

SYNOPSIS

wsh [–C textcolor,pagecolor,hilitecolor,cursorcolor]
[–C textcolor,pagecolor,hilitecolor,cursorcolor,selfg,selbg] [**–d**]
[–E] [**–f** font] [**–F**] [**–H**] [**–l** logfile] [**–L** dev1,dev2]
[–m cols,lines] [**–n** name] [**–p** x,y] [**–r** lines] [**–R** device]
[–s cols,lines] [**–t** title] [**–v**] [**–Z** number]
[–c cmd [args]]

DESCRIPTION

wsh is a terminal emulation program that runs a shell (or other UNIX command) within its own window on the screen. Each *wsh* provides menus that allow you to interactively attach and select windows, change fonts and window sizes, and create new *wsh*s.

Command line options are available to specify the font, window size, title, and position when *wsh* starts up. The command line options include:

–C *textcolor,pagecolor,hilitecolor,cursorcolor*
Set the color map indices used by *wsh* to display text characters, the background page, highlighted text (reverse video) characters, and the block cursor.

–C *textcolor,pagecolor,hilitecolor,cursorcolor,selfg,selbg*
Set the color map indices used by *wsh* to display text characters, the background page, highlighted text (reverse video) characters, the block cursor, the selection text characters, and the selection background.

–d Make *wsh* run in the foreground. This is for applications that use *wsh* as a subprocess, and need to know when *wsh* exits. Identical to the **-Z1** flag.

–E Inform *wsh* that it should hold the display when the command exits, if and only if the command exits with an error.

–f*font* Set the specified font for displaying text. For example, the following command line opens a *wsh* window using a 14-point Courier font:

```
wsh -f Courier14
```

–F Start *wsh* in fixed terminal size mode. The terminal size is fixed initially at the maximum window size specified by the –m command line option, or the default maximum size (80,40). While in fixed terminal size mode, making the window smaller will hide some of the fixed sized terminal's data. The "set size" menu may be used to set up a new fixed terminal size. By default, *wsh* starts up in variable terminal size mode. While in variable terminal size mode, reshaping the window causes the terminal itself to be resized accordingly. In this case, the terminal's data is never hidden because it is wrapped within the confines of the window itself.

—H Hold *wsh* after its child program has exited, to permit viewing its output. See –E for a more useful variation.

—l *logfile* Have *wsh* log the output of the child program into *logfile*.

—L *dev1,dev2*
Have *wsh* use a specific pty device, instead of allocating one. dev1 is the name of the pty master, dev2 is the name of the pty slave. This is normally used for the console window only.

—m *cols,lines*
Set the maximum window size, as the number of columns wide by the number of lines high. The window manager will not allow the window to be reshaped larger than the maximum window size. If the **–m** option is not given, the maximum window size is set to the initial window size, as specified by the –s command line option, or the default size. If the **–F** option is given, the maximum window size is also used as the initial fixed terminal size at startup.

—n *name* Provide a unique window name for a *wsh*. The window manager will (optionally) look up *name* in *user.ps* to find initial positioning information. If **–n** is given with no argument, the system uses the name of the command running within *wsh*. If no **–n** is given, the system does not ask *user.ps* for positioning information.

—p *x,y* Set the position of the lower-left corner of the window to the screen pixel location *x,y*. If the **–p** option (or *user.ps*, see **–n** above) does not provide a window location, the window manager prompts for it.

—r *lines* set the number of *lines* of text retained by *wsh*. Using the scrollbar one can view all of the lines retained by *wsh*. New lines entered by *wsh* are retained up to the limit specified by *lines;* after this, lines are deleted from the ''top'' of the list of lines. Setting *lines* to zero eliminates the scroll bar.

—R *device*
Provide an alternate (redirect) output device. When *wsh* receives a "toggle-redirect" command from a locally bound function key, the output of the *wsh* is redirected to the given device.

–s *cols,lines*
: Set the initial size of the window by specifying the number of columns wide by the number of lines high. The default size is 80,40. If the –s size given is larger than the default maximum window size (80,40), the effective maximum size expands to accommodate it.

–t *title* Set the title of the window to the *title*. If a null string is provided, (–t ""), *wsh* will appear without a title bar.

–v Make wsh run as a vt100 terminal emulator. Normally, wsh emulates the "iris-ansi" terminal, which is almost a vt100. This switch makes a few minor changes to the emulation for vt100 compatibility.

–Z1 Make *wsh* run in the foreground. This is for applications that use *wsh* as a subprocess, and need to know when *wsh* exits. This is identical to the –d flag.

–Z2 Tell *wsh* that it is going to run the console and to not go into the background.

–Z3 Tell *wsh* that it is going to run the console, not to go into the background, and to fork a shell.

–Z4 Normally *wsh* will change the process group of the child process (and the tty) during startup. This flag disables that behavior.

–Z5 Draw the *wsh* window without a 4Sight window frame. if you use this option, you cannot use the window manager to manipulate the window once you initially position it on the screen.

–Z6 Run *wsh* without a keyboard. Any data received from the main keyboard is ignored. Use this when you need an output only *wsh*.

–Z7 Start up *wsh* in an iconic form.

–c *cmd* [*args*]
: Execute a child program within the *wsh* window, using the specified command line arguments, rather than the default shell. The –c option will pass all trailing arguments to *wsh* , to be executed as a command line. Thus, if –c is given, it must appear as the final *wsh* option.

You can restore the size of a *wsh* window using the ''previous'' item on the window menu. This allows you to toggle back and forth between two window sizes and positions. Note that font changes which affect window size reset the previous size.

When you log on to a remote system, use vt100 if no iris-ansi entry exists in the terminfo or termcap file. You must set *wsh* terminal size to 80,24 for this to work properly. Note that certain vt100 capabilities are not present in *wsh* and thus certain foreign applications may not function properly.

When you use *vi, emacs,* or other visual-mode programs remotely, you must size the terminal window to match the terminal size referenced by the current $TERM environment variable.

wsh recognizes escape sequences as listed in the *4Sight Programmer's Guide,* Section 1, Appendix A, "Terminal and Keyboard Data".

When in alternate keypad mode (ESC =), the special function keys F9-F12 on the IRIS keyboard emulate the keys PF1-PF4 on a VT100 keyboard. (ESC P, ESC Q, ESC R, ESC S).

MENU USAGE

The *wsh* menu provides the following facilities:

select	Pop this *wsh* to the top of the window hierarchy.
copy	Copy selected text to the cut buffer. This text can be sent to a *wsh* window or to another *jot* window, as well as the window it was copied from.
send	Send the data in the cut buffer to the child program, as if it had been typed. Clicking the middle mouse will send any copied text from the cut buffer to the child process.
size	Size provides a submenu which allows control over the size of terminal that *wsh* is emulating.
font	Font provides a submenu which leads to further submenus, all of which all you to pick a new font and font size to use.
clone	Clone makes a visual duplicate of *wsh*. The duplicate will have the same size, color, and font choice as its progenitor. It will run the same command if the –c argument is given.

ADVANCED FEATURES

wsh now provides a text selection facility that allows text to be transferred from the display of one *wsh* window and sent as input to any other *wsh* window.

Select text with the left mouse button by performing the following actions:

click-hold-release	This action selects all text between the the point where the left mouse button is first clicked down and the point where it is released.

click-click	Double-clicking the left mouse causes the word over which the mouse cursor is clicked to be selected.
shift&click-hold-release	If you have already selected a block of text, you can extend your selection either forwards or backwards by holding down the shift key while clicking the left mouse.
alt&click-hold-release	Same as *click-hold-release*, except that an implicit copy (to the cut buffer) is performed when the left mouse button is released. You can use *alt&shift&click* as well.

Clicking the middle mouse will send any copied text from the cut buffer to the child process.

SCROLL BAR

wsh now has an improved scroll bar with a proportional thumb.

Clicking on the up and down arrows with the left mouse button causes the *wsh* window to scroll up or down one line, respectively. Holding down the left button over either arrow causes the scrolling to auto-repeat. Holding down the shift key while clicking on either arrow causes the window to scroll up or down one full page of text.

The scroll bar's thumb can also be used to scroll through the *wsh* window by clicking and holding down the left mouse button while dragging the mouse cursor up and down. The size of the thumb is variable, and indicates the percentage of total text stored in the window's buffer that is currently visible in the window.

KEY BINDING

wsh supports limited key binding through the *bindkey*(1) command. See the *bindkey* man page for details.

FILES

user.ps
/usr/lib/fmfonts

SEE ALSO

bindkey(1), jot(1G)
4Sight Programmer's Guide, Section 1, Chapter 5, "Using *wsh* with GL Clients", and Appendix A, "Terminal and Keyboard Data".

NAME

X - a portable, network-transparent window system

SYNOPSIS

The X Window System is a network transparent window system developed at MIT which runs on a wide range of computing and graphics machines. The core distribution from MIT has support for the following operating systems:

Ultrix 3.1 (Digital)
SunOS 4.0.3 (Sun)
HP-UX 6.5 (Hewlett-Packard)
Domain/OS 10.1 (HP/Apollo)
A/UX 1.1 (Apple)
AIX RT-2.2 and PS/2-1.1 (IBM)
AOS-4.3 (IBM)
UTEK 4.0 (Tektronix)
NEWS-OS 3.2 (Sony; client only)
UNICOS 5.0.1 (Cray; client only)
UNIX(tm) System V, Release 3.2 (AT&T 6386 WGS; client only)

It should be relatively easy to build the client-side software on a variety of other system. Commercial implementations are also available for a wide range of platforms.

The X Consortium requests that the following names be used when referring to this software:

X
X Window System
X Version 11
X Window System, Version 11
X11

X Window System is a trademark of the Massachusetts Institute of Technology.

DESCRIPTION

X Window System servers run on computers with bitmap displays. The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of the functions that are available, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, the *X Toolkit Intrinsics - C Language Interface* manual, and various toolkit documents.

The number of programs that use *X* is growing rapidly. Of particular interest are: a terminal emulator (*xterm*), a window manager (*twm*), a display manager (*xdm*), mail managing utilities (*xmh* and *xbiff*), a manual page browser (*xman*), a bitmap editor (*bitmap*), access control programs (*xauth* and *xhost*), user preference setting programs (*xrdb*, *xset*, *xsetroot*, and *xmodmap*), a load monitor (*xload*), clocks (*xclock* and *oclock*), a font displayer (*xfd*), utilities for listing information about fonts, windows, and displays (*xlsfonts*, *xfontsel*, *xlswins*, *xwininfo*, *xlsclients*, *xdpyinfo*, and *xprop*), a diagnostic for seeing what events are generated and when (*xev*), screen image manipulation utilities (*xwd*, *xwud*, *xpr*, and *xmag*), and various demos (*xeyes*, *ico*, *muncher*, *puzzle*, *xgc*, etc.).

Many other utilities, window managers, games, toolkits, etc. are available from the user-contributed software. See your site administrator for details.

STARTING UP

There are two main ways of getting the X server and an initial set of client applications started. The particular method used depends on what operating system you are running and on whether or not you use other window systems in addition to X.

xdm (the X Display Manager)

If you want to always have X running on your display, your site administrator can set your machine up to use the X Display Manager *xdm*. This program is typically started by the system at boot time and takes care of keeping the server running and getting users logged in. If you are running *xdm*, you will see a window on the screen welcoming you to the system and asking for your username and password. Simply type them in as you would at a normal terminal, pressing the Return key after each. If you make a mistake, *xdm* will display an error message and ask you to try again. After you have successfully logged in, *xdm* will start up your X environment. By default, if you have an executable file named *.xsession* in your home directory, *xdm* will treat it as a program (or shell script) to run to start up your initial clients (such as terminal emulators, clocks, a window manager, user settings for things like the background, the speed of the pointer, etc.). Your site administrator can provide details.

xinit **(run manually from the shell)**

Sites that support more than one window system might choose to use the *xinit* program for starting X manually. If this is true for your machine, your site administrator will probably have provided a program named "x11", "startx", or "xstart" that will do site-specific initialization (such as loading convenient default resources, running a window manager, displaying a clock, and starting several terminal emulators) in a nice way. If not, you can build such a script using the *xinit* program. This utility simply runs one user-specified program to start the server, runs another to start up any desired clients, and then waits for either to finish. Since either or both of the user-specified programs may be a shell script, this gives substantial flexibility at the expense of a nice interface. For this reason, *xinit* is not intended for end users.

DISPLAY NAMES

From the user's prospective, every X server has a *display name* of the form:

hostname:displaynumber.screennumber

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

hostname

The *hostname* specifies the name of the machine to which the display is physically connected. If the hostname is not given, the most efficient way of communicating to a server on the same machine will be used.

displaynumber

The phrase "display" is usually used to refer to collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most workstations tend to only have one keyboard, and therefore, only one display. Larger, multi-user systems, however, will frequently have several displays so that more than one person can be doing graphics work at once. To avoid confusion, each display on a machine is assigned a *display number* (beginning at 0) when the X server for that display is started. The display number must always be given in a display name.

screennumber

Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a *screen number* (beginning at 0) when the X server for that display is started. If the screen number is not given, then screen 0 will be used.

On POSIX systems, the default display name is stored in your DISPLAY environment variable. This variable is set automatically by the *xterm* terminal emulator. However, when you log into another machine on a network, you'll need to set DISPLAY by hand to point to your display. For example,

```
% setenv DISPLAY myws:0
$ DISPLAY=myws:0; export DISPLAY
```

Finally, most X programs accept a command line option of **-display** *displayname* to temporarily override the contents of DISPLAY. This is most commonly used to pop windows on another person's screen or as part of a "remote shell" command to start an xterm pointing back to your display. For example,

```
% xeyes -display joesws:0 -geometry 1000x1000+0+0
% rsh big xterm -display myws:0 -ls </dev/null &
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, The *hostname* part of the display name is used to determine the type of channel (also called a transport layer) to be used. The sample servers from MIT support the following types of connections:

local

The hostname part of the display name should be the empty string. For example: *:0*, *:1*, and *:0.1*. The most efficient local transport will be chosen.

TCP/IP

The hostname part of the display name should be the server machine's IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example: *expo.lcs.mit.edu:0*, *expo:0*, *18.30.0.212:0*, *bigmachine:1*, and *hydra:0.1*.

DECnet

The hostname part of the display name should be the server machine's nodename followed by two colons instead of one. For example: *myws::0*, *big::1*, and *hydra::0.1*.

ACCESS CONTROL

The sample server provides two types of access control: an authorization protocol which provides a list of "magic cookies" clients can send to request access, and a list of hosts from which connections are always accepted. *Xdm* initializes magic cookies in the server, and also places them in a file accessible to the user. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients with are explicitly authorized can connect to the display. When you add entries to the host list (with *xhost*), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file for authorization which both *xdm* and *Xlib* use can be specified with the environment variable **XAUTHORITY**, and defaults to the file **.Xauthority** in the home directory. *Xdm* uses **$HOME/.Xauthority** and will create it or merge in authorization records if it already exists when a user logs in.

To manage a collection of authorization files containing a collection of authorization records use *xauth*. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you login. As the files are machine-independent, you can also simply copy the files or use NFS to share them. If you use several machines, and share a common home directory with NFS, then you never really have to worry about authorization files, the system should work correctly by default. Note that magic cookies transmitted ''in the clear'' over NFS or using *ftp* or *rcp* can be ''stolen'' by a network eavesdropper, and as such may enable unauthorized access. In many environments this level of security is not a concern, but if it is, you need to know the exact semantics of the particular magic cookie to know if this is actually a problem.

GEOMETRY SPECIFICATIONS

One of the advantages of using window systems instead of hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running (described below), most X programs accept a command line argument of the form **-geometry** *WIDTHxHEIGHT+XOFF+YOFF* (where *WIDTH*, *HEIGHT*, *XOFF*, and *YOFF* are numbers) for specifying a preferred size and location for this application's main window.

The *WIDTH* and *HEIGHT* parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The *XOFF* and *YOFF* parts are measured in pixels and are used to specify the distance of the window from the left or right and top and bottom edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window. The X offset may be specified in the following ways:

+XOFF	The left edge of the window is to be placed *XOFF* pixels in from the left edge of the screen (i.e. the X coordinate of the window's origin will be *XOFF*). *XOFF* may be negative, in which case the window's left edge will be off the screen.
-XOFF	The right edge of the window is to be placed *XOFF* pixels in from the right edge of the screen. *XOFF* may be negative, in which case the window's right edge will be off the screen.

The Y offset has similar meanings:

+YOFF	The top edge of the window is to be *YOFF* pixels below the top edge of the screen (i.e. the Y coordinate of the window's origin will be *YOFF*). *YOFF* may be negative, in which case the window's top edge will be off the screen.
-YOFF	The bottom edge of the window is to be *YOFF* pixels above the bottom edge of the screen. *YOFF* may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either *XOFF* or *YOFF* both must be present. Windows can be placed in the four corners of the screen using the following specifications:

+0+0	upper left hand corner.
-0+0	upper right hand corner.
-0-0	lower right hand corner.
+0-0	lower left hand corner.

In the following examples, a terminal emulator will be placed in roughly the center of the screen and a load average monitor, mailbox, and clock will be placed in the upper right hand corner:

```
xterm -fn 6x10 -geometry 80x24+30+200 &
xclock -geometry 48x48-0+0 &
xload -geometry 48x48-96+0 &
xbiff -geometry 48x48-48+0 &
```

WINDOW MANAGERS

The layout of windows on the screen is controlled by special programs called *window managers*. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The core distribution comes with a window manager named *twm* which supports overlapping windows, popup menus, point-and-click or click-to-type input models, title bars, nice icons (and an icon manager for those who don't like separate icon windows).

Several other window managers are available in the user-contributed software: *gwm*, *m_swm*, *olwm*, and *tekwm*.

FONT NAMES

Collections of characters for displaying text and symbols in X are known as *fonts*. A font typically contains images that share a common appearance and look nice together (for example, a single size, boldness, slant, and character set). Similarly, collections of fonts that are based on a common type face (the variations are usually called roman, bold, italic, bold italic, oblique, and bold oblique) are called *families*.

Sets of font families of the same resolution (usually measured in dots per inch) are further grouped into *directories* (so named because they were initially stored in file system directories). Each directory contains a database which lists the name of the font and information on how to find the font. The server uses these databases to translate *font names* (which have nothing to do with file names) into font data.

The list of font directories in which the server looks when trying to find a font is controlled by the *font path*. Although most installations will choose to have the server start up with all of the commonly used font directories, the font path can be changed at any time with the *xset* program. However, it is important to remember that the directory names are on the server's machine, not on the application's.

The default font path for the sample server contains three directories:

/usr/lib/X11/fonts/misc
: This directory contains many miscellaneous fonts that are useful on all systems. It contains a small family of fixed-width fonts in pixel heights 5 through 10, a family of fixed-width fonts from Dale Schumacher in similar pixel heights, several Kana fonts from Sony Corporation, a Kanji font, the standard cursor font, two cursor fonts from Digital Equipment Corporation, and OPEN LOOK(tm) cursor and glyph fonts from Sun Microsystems. It also has font name aliases for the font names **fixed** and **variable**.

/usr/lib/X11/fonts/75dpi
: This directory contains fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc. for 75 dots per inch

displays. An integrated selection of sizes, styles, and weights are provided for each family.

/usr/lib/X11/fonts/100dpi
This directory contains 100 dots per inch versions of some of the fonts in the *75dpi* directory.

Font databases are created by running the *mkfontdir* program in the directory containing the source or compiled versions of the fonts (in both compressed and uncompressed formats). Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the *xset* program. For example, to add a font to a private directory, the following commands could be used:

```
% cp newfont.snf ~/myfonts
% mkfontdir ~/myfonts
% xset fp rehash
```

The *xlsfonts* program can be used to list all of the fonts that are found in font databases in the current font path. Font names tend to be fairly long as they contain all of the information needed to uniquely identify individual fonts. However, the sample server supports wildcarding of font names, so the full specification

-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1

could be abbreviated as:

--courier-medium-r-normal--*-100-*-*-*-*-*-**

or, more tersely (but less accurately):

-courier-medium-r-normal--*-100-

Because the shell also has special meanings for * and ?, wildcarded font names should be quoted:

```
% xlsfonts -fn '*-courier-medium-r-normal--*-100-*'
```

If more than one font in a given directory in the font path matches a wildcarded font name, the choice of which particular font to return is left to the server. However, if fonts from more than one directory match a name, the returned font will always be from the first such directory in the font path. The example given above will match fonts in both the *75dpi* and *100dpi* directories; if the *75dpi* directory is ahead of the *100dpi* directory in the font path, the smaller version of the font will be used.

COLOR NAMES

Most applications provide ways of tailoring (usually through resources or command line arguments) the colors of various elements in the text and graphics they display. Although black and white displays don't provide much of a choice, color displays frequently allow anywhere between 16 and 16 million different colors.

Colors are usually specified by their commonly-used names (for example, *red*, *white*, or *medium slate blue*). The server translates these names into appropriate screen colors using a color database that can usually be found in */usr/lib/X11/rgb.txt*. Color names are case-insensitive, meaning that *red*, *Red*, and *RED* all refer to the same color.

Many applications also accept color specifications of the following form:

```
#rgb
#rrggbb
#rrrgggbbb
#rrrrggggbbbb
```

where *r*, *g*, and *b* are hexadecimal numbers indicating how much *red*, *green*, and *blue* should be displayed (zero being none and ffff being on full). Each field in the specification must have the same number of digits (e.g., #rrgb or #gbb are not allowed). Fields that have fewer than four digits (e.g. #rgb) are padded out with zero's following each digit (e.g. #r000g000b000). The eight primary colors can be represented as:

black	#000000000000 (no color at all)
red	#ffff00000000
green	#0000ffff0000
blue	#00000000ffff
yellow	#ffffffff0000 (full red and green, no blue)
magenta	#ffff0000ffff
cyan	#0000ffffffff
white	#ffffffffffff (full red, green, and blue)

Unfortunately, RGB color specifications are highly unportable since different monitors produce different shades when given the same inputs. Similarly, color names aren't portable because there is no standard naming scheme and because the color database needs to be tuned for each monitor.

Application developers should take care to make their colors tailorable.

KEYS

The X keyboard model is broken into two layers: server-specific codes (called *keycodes*) which represent the physical keys, and server-independent symbols (called *keysyms*) which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

modifier list

Some keys (such as Shift, Control, and Caps Lock) are known as *modifier* and are used to select different symbols that are attached to a single key (such as Shift-a generates a capital A, and Control-l generates a formfeed character ^L). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is pressed or released, the server generates an *event* that contains the keycode of the indicated key as well as a mask that specifies which of the modifier keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift lock keys on the keyboard.

keymap table

Applications translate event keycodes and modifier masks into keysyms using a *keysym table* which contains one row for each keycode and one column for various modifier states. This table is initialized by the server to correspond to normal typewriter conventions, but is only used by client programs.

Although most programs deal with keysyms directly (such as those written with the X Toolkit Intrinsics), most programming libraries provide routines for converting keysyms into the appropriate type of string (such as ISO Latin-1).

OPTIONS

Most X programs attempt to use the same names for command line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

-display *display*

This option specifies the name of the X server to use.

-geometry *geometry*

This option specifies the initial size and location of the window.

-bg *color*, **-background** *color*

Either option specifies the color to use for the window background.

–bd *color*, **–bordercolor** *color*
Either option specifies the color to use for the window border.

–bw *number*, **–borderwidth** *number*
Either option specifies the width in pixels of the window border.

–fg *color*, **–foreground** *color*
Either option specifies the color to use for text or graphics.

–fn *font*, **-font** *font*
Either option specifies the font to use for displaying text.

–iconic
This option indicates that the user would prefer that the application's windows initially not be visible as if the windows had be immediately iconified by the user. Window managers may choose not to honor the application's request.

–name
This option specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.

–rv, –reverse
Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv
This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

–selectionTimeout
This option specifies the timeout in milliseconds within which two communicating applications must respond to one another for a selection request.

–synchronous
This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

−title *string*
This option specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.

−xnllanguage *language[_territory][.codeset]*
This option specifies the language, territory, and codeset for use in resolving resource and other filenames.

−xrm *resourcestring*
This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings of the form

*appname*subname*subsubname...: value*

that are read in from various places when an application is run. By convention, the application name is the same as the program name, but with the first letter capitalized (e.g. *Bitmap* or *Emacs*) although some programs that begin with the letter "x" also capitalize the second letter for historical reasons. The precise syntax for resources is:

```
ResourceLine        =Comment | ResourceSpec
Comment             ="!" string | <empty line>
ResourceSpec        =WhiteSpace ResourceName WhiteSpace ":" WhiteSpace value
ResourceName        =[Binding] ComponentName {Binding ComponentName}
Binding             ="." | "*"
WhiteSpace          ={" " | "\t"}
ComponentName       ={"a"-"z" | "A"-"Z" | "0"-"9" | "_" | "-"}
value               =string
string              ={<any character not including "\n">}
```

Note that elements enclosed in curly braces ({...}) indicate zero or more occurrences of the enclosed elements

To allow values to contain arbitrary octets, the 4-character sequence *nnn*, where n is a digit in the range of "0"–"7", is recognized and replaced with a single byte that contains this sequence interpreted as an octal number. For example, a value containing a NULL byte can be stored by specifying "\000".

The *Xlib* routine *XGetDefault(3X)* and the resource utilities within Xlib and the X Toolkit Intrinsics obtain resources from the following sources:

RESOURCE_MANAGER root window property
: Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window using the *xrdb* program. This is frequently taken care of when the user starts up X through the display manager or *xinit*.

application-specific files
: Programs that use the X Toolkit Intrinsics will also look in the directories named by the environment variable XUSERFILESEARCHPATH or the environment variable XAPPLRESDIR, plus directories in a standard place (usually under /usr/lib/X11/, but this can be overridden with the XFILESEARCHPATH environment variable) for application-specific resources. See the *X Toolkit Intrinsics - C Language Interface* manual for details.

XENVIRONMENT
: Any user- and machine-specific resources may be specified by setting the XENVIRONMENT environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, a file named *$HOME/*.Xdefaults-*hostname* is looked for instead, where *hostname* is the name of the host where the application is executing.

–xrm *resourcestring*
: Applications that use the X Toolkit Intrinsics can have resources specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **–xrm** arguments may be given on the command line.

Program resources are organized into groups called *classes*, so that collections of individual resources (each of which are called *instances*) can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics will have at least the following resources:

background (class Background)
This resource specifies the color to use for the window background.

borderWidth (class BorderWidth)
This resource specifies the width in pixels of the window border.

borderColor (class BorderColor)
This resource specifies the color to use for the window border.

Most applications using the X Toolkit Intrinsics also have the resource **foreground** (class Foreground), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set Background and Foreground classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources. For example,

```
bitmap*Dashed:  off
XTerm*cursorColor:  gold
XTerm*multiScroll:  on
XTerm*jumpScroll:  on
XTerm*reverseWrap:  on
XTerm*curses:  on
XTerm*Font:  6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime: 500
XTerm*charClass:  33:48,37:48,45-47:48,64:48
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit:  on
XTerm*ttyModes:  intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: red
XLoad*highlight: black
XLoad*borderWidth: 0
emacs*Geometry:  80x65-0-0
emacs*Background:  #5b7686
emacs*Foreground:  white
emacs*Cursor:  white
emacs*BorderColor:  white
emacs*Font:  6x10
xmag*geometry:  -0-0
```

```
xmag*borderColor: white
```

If these resources were stored in a file called *.Xresources* in your home directory, they could be added to any existing resources in the server with the following command:

```
% xrdb -merge $HOME/.Xresources
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of automatically loading resources. See the *Xlib* manual section *Using the Resource Manager* for more information.

EXAMPLES

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command's manual page.

```
% xrdb -load $HOME/.Xresources
% xmodmap -e "keysym BackSpace = Delete"
% mkfontdir /usr/local/lib/X11/otherfonts
% xset fp+ /usr/local/lib/X11/otherfonts
% xmodmap $HOME/.keymap.km
% xsetroot -solid '#888'
% xset b 100 400 c 50 s 1800 r on
% xset q
% twm
% xmag
% xclock -geometry 48x48-0+0 -bg blue -fg white
% xeyes -geometry 48x48-48+0
% xbiff -update 20
% xlsfonts '*helvetica*'
% xlswins -l
% xwininfo -root
% xdpyinfo -display joesworkstation:0
% xhost -joesworkstation
% xrefresh
% xwd | xwud
% bitmap companylogo.bm 32x32
% xcalc -bg blue -fg magenta
% xterm -geometry 80x66-0-0 -name myxterm $*
```

DIAGNOSTICS

A wide variety of error messages are generated from various programs. Various toolkits are encouraged to provide a common mechanism for locating error text so that applications can be tailored easily. Programs written to interface directly to the *Xlib* C language library are expected to do their own error checking.

The default error handler in *Xlib* (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in */usr/lib/X11/XErrorDB*. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsics encounter errors converting resource strings to the appropriate internal format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (e.g. color vs. monochrome, lots of fonts vs. very few, etc.), although it can pose problems for trying to determine why an application might be failing. This behavior can be overridden by the setting the *StringConversionsWarning* resource.

To force the X Toolkit Intrinsics to always print string conversion error messages, the following resource should be placed at the top of the file that gets loaded onto the RESOURCE_MANAGER property using the *xrdb* program (frequently called *.Xresources* or *.Xres* in the user's home directory):

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

BUGS

If you encounter a repeatable bug, please contact your site administrator for instructions on how to submit an X Bug Report.

SEE ALSO

XConsortium(1), XStandards(1), appres(1), bdftosnf(1), bitmap(1), imake(1), listres(1), maze(1), mkfontdir(1), muncher(1), oclock(1), puzzle(1), resize(1), showsnf(1), twm(1), xauth(1), xbiff(1), xcalc(1), xclipboard(1), xclock(1), xditview(1), xdm(1), xdpyinfo(1), xedit(1), xev(1), xeyes(1), xfd(1), xfontsel(1), xhost(1), xinit(1), xkill(1), xload(1), xlogo(1), xlsatoms(1), xlsclients(1), xlsfonts(1), xlswins(1), xmag(1), xman(1), xmh(1), xmodmap(1), xpr(1), xprop(1), xrdb(1), xrefresh(1), xset(1), xsetroot(1), xstdcmap(1), xterm(1), xwd(1), xwininfo(1), xwud(1), Xserver(1), Xapollo(1), Xcfbpmax(1), Xhp(1), Xibm(1), XmacII(1), Xmfbpmax(1), Xqdss(1), Xqvss(1), Xsun(1), Xtek(1), kbd_mode(1), todm(1), tox(1), biff(1), init(8), ttys(5), *Xlib – C Language X Interface*, *X Toolkit Intrinsics - C Language Interface*, and *Using and Specifying X Resources*

COPYRIGHT

The following copyright and permission notice outlines the rights and restrictions covering most parts of the core distribution of the X Window System from MIT. Other parts have additional or different copyrights and permissions; see the individual source files.

Copyright 1984, 1985, 1986, 1987, 1988, and 1989, by the Massachusetts Institute of Technology.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. MIT makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

TRADEMARKS

UNIX and OPEN LOOK are trademarks of AT&T. X Window System is a trademark of MIT.

AUTHORS

A cast of thousands, literally. The X distribution is brought to you by the MIT X Consortium. The staff members at MIT responsible for this release are: Donna Converse (MIT X Consortium), Jim Fulton (MIT X Consortium), Michelle Leger (MIT X Consortium), Keith Packard (MIT X Consortium), Chris Peterson (MIT X Consortium), Bob Scheifler (MIT X Consortium), and Ralph Swick (Digital/MIT Project Athena).

NAME

x10tox11 – X version 10 to version 11 protocol converter

SYNOPSIS

x10tox11 [-display host:display]

DESCRIPTION

x10tox11 masquerades as an X Window System Version 10 server. It enables an X Version 10 client to run unchanged under X Version 11 by converting Version 10 requests into appropriate Version 11 requests, and by converting all Version 11 events received from the server into Version 10 events. From the perspective of Version 10 clients, all Version 11 clients look like Version 10 clients; and from the perspective of Version 11 clients, all Version 10 clients just look like Version 11 clients. Hence, a Version 11 window manager can manipulate Version 10 clients.

This program does NOT use the X10 *libnest* ddX library. It does actual protocol translation, rather than simply using X11 graphics calls to implement X10 low level operations. As a result, it is both faster and more robust than the X10 Xnest server.

TYPICAL USAGE

The protocol converter must be run after the X11 server is running and should be run in the background:

x10tox11 &

The program will continue to run until you intentionally kill it or the X11 server is shut down.

OPTIONS

-display host:display

Standard option for specifying the X11 display to which you wish to be connected. By default, it uses unix:0.0. Note that *x10tox11* will always pretend to be an X10 server with the same display number as the X11 server to which it connects. For example, if the DISPLAY environment variable or the *-display* option specifies *fizzle:1.0*, then *x10tox11* will connect to the X11 server on host *fizzle* for display 1 and then will pretend to the the X10 server for display 1. Consequently, your X10 clients will expect to have the environment variable DISPLAY set to *fizzle:1* (but they should still work even if your X10 clients use *fizzle:1.0*).

MinimumTileSize=n

Set minimum acceptable tile size to *n*. There is a difference in semantics between X10's XQueryShape and X11's XQueryBestSize such that X11 will allow any tile size but will return the optimum whereas X10 enforced a minimum tile size. Usually this minimum tile size was 16 and this is the default for *x10tox11*. If you find that this makes your X10 clients break, then you can override it with this option.

help

This prints out a usage message and exits.

NoOverrideRedirect

This instructs *x10tox11* to make every effort not to use OverrideRedirect when creating and mapping windows. Normally, *x10tox11* creates all windows with the OverrideRedirect attribute set to true. Placing this option on the command line will cause *x10tox11* not to use OverrideRedirect except for windows that look like they might be menus. This will allow window managers that provide title bars to do so. Unfortunately, it is impossible to determine ahead of time what an X10 client intends to do with windows. In addition, X10 clients are known to spontaneously unmap their windows which upsets X11 window managers unless the OverrideRedirect attribute is true. Further, some X11 window managers may refuse to resize or move windows that are marked with OverrideRedirect. This may can be fixed to some extent when an Inter Client Communications Convention Manual (ICCCM) is adopted by the X11 community.

SEE ALSO

X(1), Xserver(1)

BUGS

There are limitations with respect to emulating Version 10 through a Version 11 server. See the file /usr/lib/X/x10tox11.help for more details.

Some window managers may refuse to move, resize or perform any operations on X10 client windows because, by default,

If the source is compiled with certain flags, there are significant debugging facilities available. Using the *help* option will tell you whether debugging facilities are available. *x10tox11* marks them with OverrideRedirect. See OPTIONS above.

COPYRIGHT

AUTHOR

Todd Brunhoff, Visual Systems Laboratory, Tektronix.

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **–i** flag). Flags **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **–l** vs. **–n**), the last flag has precedence. *Flag* values are:

–l*number* *command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **–x** is forced.

–i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. **{}** is assumed for *replstr* if not specified.

−n*number*	Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.
−t	Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
−p	Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.
−x	Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.
−s*size*	The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If −s is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
−e*eofstr*	*eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if **−e** is not coded. The value **−e** with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *xargs* reads standard input until either end-of-file or the logical

EOF string is encountered.

xargs will terminate if either it receives a return code of −1 from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with −1.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs −i −t mv $1/{ } $2/{ }
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs −p −l ar r arch
2. ls | xargs −p −l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs −n2 diff
```

SEE ALSO

sh(1).

NAME

xauth - X authority file utility

SYNOPSIS

xauth [-f *authfile*] [-vqib] [*command arg...*]

DESCRIPTION

The *xauth* program is used to edit and display the authorization information used in connecting to the X server. This program is usually to extract authorization records from one machine and merge them in on another (as is the case when using remote logins or to grant access to other users). Commands (described below) may be entered interactively, on the *xauth* command line, or in scripts. Note that this program does not contact the X server.

OPTIONS

The following options may be used with *xauth*. They may be given indivually (e.g. *–q –i*) or may combined (e.g. *-qi*):

–f *authfile*
This option specifies the name of the authority file to use. By default, *xauth* will use the file specified by the XAUTHORITY environment variable or *Xauthority* in the user's home directory.

–q
This option indicates that *xauth* should operate quietly and not print unsolicited status messages. This is the default if an *xauth* command is is given on the command line or if the standard output is not directed to a terminal.

–v
This option indicates that *xauth* should operate verbosely and print status messages indicating the results of various operations (e.g. how many records have been read in or written out). This is the default if *xauth* is reading commands from its standard input and its standard output is directed to a terminal.

–i
This option indicates that *xauth* should ignore any authority file locks. Normally, *xauth* will refuse to read or edit any authority files that have been locked by other programs (usually *xdm* or another *xauth*).

–b
This option indicates that *xauth* should attempt to break any authority file locks before proceeding and should only be used to clean up stale locks.

COMMANDS

The following commands may be used to manipulate authority files:

add *displayname protocolname hexkey*
: An authorization entry for the indicated display using the given protocol and key data is added to the authorization file. The data is specified as an even-lengthed string of hexidecimal digits, each pair representing one octet. The first digit gives the most significant 4 bits of the octet and the second digit gives the least significant 4 bits. A protocol name consisting of just a single period is treated as an abbreviation for *MIT-MAGIC-COOKIE-1*.

[n]extract *filename displayname...*
: Authorization entries for each of the specified displays are written to the indicated file. If the *nextract* command is used, the entries are written in a numeric format suitable for non-binary transmission (such as secure electronic mail). The extracted entries can be read back in using the *merge* and *nmerge* commands. If the the filename consists of just a single dash, the entries will be written to the standard output.

[n]list [*displayname...*]
: Authorization entries for each of the specified displays (or all if no displays are named) are printed on the standard output. If the *nlist* command is used, entries will be shown in the numeric format used by the *nextract* command; otherwise, they are shown in a textual format. Key data is always displayed in the hexidecimal format given in the description of the *add* command.

[n]merge [*filename...*]
: Authorization entries are read from the specified files and are merged into the authorization database, superceeding any matching existing entries. If the *nmerge* command is used, the numeric format given in the description of the *extract* command is used. If a filename consists of just a single dash, the standard input will be read if it hasn't been read before.

remove *displayname...*
: Authorization entries matching the specified displays are removed from the authority file.

source *filename*
: The specified file is treated as a script containing *xauth* commands to execute. Blank lines and lines beginning with a sharp sign (#) are ignored. A single dash may be used to indicate the standard input, if it hasn't already been read.

info Information describing the authorization file, whether or not any changes have been made, and from where *xauth* commands are being read is printed on the standard output.

exit If any modifications have been made, the authority file is written out (if allowed), and the program exits. An end of file is treated as an implicit *exit* command.

quit The program exits, ignoring any modifications. This may also be accomplished by pressing the interrupt character.

help [*string*]
A description of all commands that begin with the given string (or all commands if no string is given) is printed on the standard output.

? A short list of the valid commands is printed on the standard output.

DISPLAY NAMES

Display names for the *add*, *[n]extract*, *[n]list*, *[n]merge*, and *remove* commands use the same format as the DISPLAY environment variable and the common *-display* command line argument. Display-specific information (such as the screen number) is unnecessary and will be ignored. Same-machine connections (such as UNIX-domain sockets, shared memory, and the Internet Protocol hostname *localhost*) are refered to as *hostname*/unix:*displaynumber* so that local entries for different machines may be stored in one authority file.

EXAMPLE

The most common use for *xauth* is to extract the entry for the current display, copy it to another machine, and merge it into the user's authority file on the remote machine:

```
% xauth extract - $DISPLAY | rsh other xauth merge -
```

ENVIRONMENT

This *xauth* program uses the following environment variables:

XAUTHORITY
to get the name of the authority file to use if the *–f* option isn't used. If this variable is not set, *xauth* will use *Xauthority* in the user's home directory.

HOME to get the user's home directory if XAUTHORITY isn't defined.

BUGS

Users that have unsecure networks should take care to use encrypted file transfer mechanisms to copy authorization entries between machines. Similarly, the *MIT-MAGIC-COOKIE-1* protocol is not very useful in unsecure environments. Sites that are interested in additional security may need to use encrypted authorization mechanisms such as Kerberos.

Spaces are currently not allowed in the protocol name. Quoting could be added for the truly perverse.

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xbiff – mailbox flag for X

SYNOPSIS

xbiff [-*toolkitoption* ...] [-option ...]

DESCRIPTION

The *xbiff* program displays a little image of a mailbox. When there is no mail, the flag on the mailbox is down. When mail arrives, the flag goes up and the mailbox beeps. By default, pressing any mouse button in the image forces *xbiff* to remember the current size of the mail file as being the "empty" size and to lower the flag.

This program is nothing more than a wrapper around the Athena *Mailbox* widget.

OPTIONS

Xbiff accepts all of the standard X Toolkit command line options along with the additional options listed below:

–help This option indicates that a brief summary of the allowed options should be printed on the standard error.

–update *seconds*
This option specifies the frequency in seconds at which *xbiff* should update its display. If the mailbox is obscured and then exposed, it will be updated immediately. The default is 60 seconds.

–file *filename*
This option specifies the name of the file which should be monitored. By default, it watches /usr/spool/mail/*username*, where *username* is your login name.

–volume *percentage*
This option specifies how loud the bell should be rung when new mail comes in.

–shape This option indicates that the mailbox window should be shaped if masks for the empty or full images are given.

The following standard X Toolkit command line arguments are commonly used with *xbiff:*

–display *display*
This option specifies the X server to contact.

–geometry *geometry*
This option specifies the prefered size and position of the mailbox window. The mailbox is 48 pixels wide and 48 pixels high and will be centered in the window.

–bg *color*
This option specifies the color to use for the background of the window. The default is "white."

–bd *color*
This option specifies the color to use for the border of the window. The default is "black."

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color*
This option specifies the color to use for the foreground of the window. The default is "black."

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Mailbox* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

checkCommand (class **CheckCommand**)
Specifies a shell command to be executed to check for new mail rather than examining the size of file. The specified string value is used as the argument to a *system*(3) call and may therefore contain i/o redirection. An exit status of 0 indicates that new mail is waiting, 1 indicates that there has been no change in size, and 2 indicates that the mail has been cleared.

file (class **File**)
Specifies the name of the file to monitor. The default is to watch /usr/spool/mail/*username*, where *username* is your login name.

onceOnly (class **Boolean**)
Specifies that the bell is only rung the first time new mail is found and is not rung again until at least one interval has passed with no mail waiting. The window will continue to indicate the presence of new mail until it has been retrieved.

width (class **Width**)
Specifies the width of the mailbox.

height (class **Height**)
Specifies the height of the mailbox.

update (class **Interval**)
Specifies the frequency in seconds at which the mail should be checked.

volume (class **Volume**)
Specifies how load the bell should be rung. The default is 33 percent.

foreground (class **Foreground**)
Specifies the color for the foreground. The default is ''black'' since the core default for background is ''white.''

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

flip (class **Flip**)
Specifies whether or not the image that is shown when mail has arrived should be inverted. The default is ''true.''

fullPixmap (class **Pixmap**)
Specifies a bitmap to be shown when new mail has arrived.

emptyPixmap (class **Pixmap**)
Specifies a bitmap to be shown when no new mail is present.

shapeWindow (class **ShapeWindow**)
Specifies whether or not the mailbox window should be shaped to the given fullPixmapMask and emptyPixmapMask.

fullPixmapMask (class **PixmapMask**)
Specifies a mask for the bitmap to be shown when new mail has arrived.

emptyPixmapMask (class **PixmapMask**)
Specifies a mask for the bitmap to be shown when no new mail is present.

ACTIONS

The *Mailbox* widget provides the following actions for use in event translations:

check() This action causes the widget to check for new mail and display the flag appropriately.

unset() This action causes the widget to lower the flag until new mail comes in.

set() This action causes the widget to raise the flag until the user resets it.

The default translation is

```
<ButtonPress>: unset()
```

ENVIRONMENT

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1), xrdb(1), stat(2)

BUGS

The mailbox bitmaps are ugly.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium
Additional hacks by Ralph Swick, DEC/MIT Project Athena

NAME

xcalc – scientific calculator for X

SYNOPSIS

xcalc [-stipple] [-rpn] [-*toolkitoption...*]

DESCRIPTION

Xcalc is a scientific calculator desktop accessory that can emulate a TI-30 or an HP-10C.

OPTIONS

xcalc accepts all of the standard toolkit command line options along with the additional options listed below:

–stipple This option indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays this makes for a nicer display.

–rpn This option indicates that Reverse Polish Notation should be used. In this mode the calculator will look and behave like an HP-10C. Without this flag, it will emulate a TI-30.

OPERATION

Pointer Usage: Most operations are done with the Button1 (usually leftmost button on the pointer). The only exception is that pressing the AC key on the TI calculator or the ON key on the HP calculator with Button3 (usually on the right) will exit the calculator.

Key Usage (Normal mode): The number keys, the +/- key, and the +, -, *, /, and = keys all do exactly what you would expect them to. It should be noted that the operators obey the standard rules of precedence. Thus, entering "3+4*5=" results in "23", not "35". The parentheses can be used to override this. For example, "(1+2+3)*(4+5+6)=" results in "6*15=90". The non-obvious keys are detailed below.

1/x replaces the number in the display with its reciprocal.

x^2 squares the number in the display.

SQRT takes the square root of the number in the display.

CE/C when pressed once, clears the number in the display without clearing the state of the machine. Allows you to re-enter a number if you screw it up. Pressing it twice clears the state, also.

AC clears everything, the display, the state, the memory, everything. Pressing it with the right button 'turns off' the calculator, in that it exits the program. Somewhat more equivalent to throwing the calculator in the trash, if we were to pursue the analogy.

INV inverts the meaning of the function keys. See the individual function keys for details.

sin computes the sine of the number in the display, as interpreted by the current DRG mode (see DRG, below). If inverted, it computes the arcsine.

cos computes the cosine, or arccosine when inverted.

tan computes the tangent, or arctangent when inverted.

DRG changes the DRG mode, as indicated by 'DEG', 'RAD', or 'GRAD' at the bottom of number window of the calculator. When in 'DEG' mode, numbers in the display are taken as being degrees. In 'RAD' mode, numbers are in radians, and in 'GRAD' mode, numbers are in gradians. When inverted, the DRG key has the nifty feature of converting degrees to radians to gradians and vice-versa. Example: put the calculator into 'DEG' mode, and type "45 INV DRG". The display should now show something along the lines of ".785398", which is 45 degrees converted to radians.

e the constant 'e'. (2.7182818...)

EE used for entering exponential numbers. For example, to enter "-2.3E-4" you'd type "2 . 3 +/- EE 4 +/-"

log calculates the log (base 10) of the number in the display. When inverted, it raises "10.0" to the number in the display. For example, typing "3 INV log" should result in "1000".

ln calcuates the log (base e) of the number in the display. When inverted, it raises "e" to the number in the display. For example, typing "e ln" should result in "1"

y^x raises the number on the left to the power of the number on the right. For example "2 y^x 3 =" results in "8", which is 2^3. For a further example, "(1+2+3) y^x (1+2) =" equals "6 y^x 3" which equals "216".

PI the constant 'pi'. (3.1415927....)

x! computes the factorial of the number in the display. The number in the display must be an integer in the range 0-500, though, depending on your math library, it might overflow long before that.

STO copies the number in the display to the memory location.

RCL copies the number from the memory location to the display.

SUM adds the number in the display to the number in the memory location.

EXC swaps the number in the display with the number in the memory location.

Key Usage (RPN mode): The number keys, CHS (change sign), +, -, *, /, and ENTR keys all do exactly what you would expect them to do. Many of the remaining keys are the same as in normal mode. The differences are detailed below.

<- is a backspace key that can be used while typing a number. It will erase digits from the display. Inverse backspace will clear the X register.

ON clears everything, the display, the state, the memory, everything. Pressing it with the right button 'turns off' the calculator, in that it exits the program. Somewhat more equivalent to throwing the calculator in the trash, if we were to pursue the analogy.

INV inverts the meaning of the function keys. This would be the "f" key on an HP calculator, but xcalc does not have the resolution to display multiple legends on each key. See the individual function keys for details.

10^x raises "10.0" to the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display.

e^x raises "e" to the number in the top of the stack. When inverted, it calcuates the log (base e) of the number in the display.

STO copies the number in the top of the stack to a memory location. There are 10 memory locations. The desired memory is specified by following this key with pressing a digit key.

RCL pushes the number from the specified memory location onto the stack.

SUM adds the number on top of the stack to the number in the specified memory location.

x:y exchanges the numbers in the top two stack positions, the X and Y registers.

R v rolls the stack downward. When inverted, it rolls the stack upward.

blank these keys were used for programming functions on the HP11-C. Their functionality has not been duplicated here.

KEYBOARD EQUIVALENTS

If you have the pointer in the xcalc window, you can use the keyboard to speed entry, as almost all of the calculator keys have a keyboard equivalent. The number keys, the operator keys, and the parentheses all have the obvious equivalent. The less-obvious equivalents are as follows:

n: +/-	!: x!
p: PI	e: EE
l: ln	^: y^x
i: INV	s: sin
c: cos	t: tan
d: DRG	BS, DEL: CE/C ("<-" in RPN mode)
CR: ENTR	q: quit

COLOR USAGE

Xcalc uses a lot of colors, given the opportunity. In the default case, it will just use two colors (Foreground and Background) for everything. This works out nicely. However, if you're a color fanatic you can specify the colors used for the number keys, the operator (+-*/=) keys, the function keys, the display, and the icon.

X DEFAULTS

stipple Indicates that the background should be stippled. The default is "on" for monochrome displays, and "off" for color displays.

rpn Specifies that the rpn mode should be used. The default is TI mode.

SEE ALSO

X(1), xrdb(1)

BUGS

HP mode may or may not work correctly.

COPYRIGHT

AUTHORS

John Bradley, University of Pennsylvania
Mark Rosenstein, MIT Project Athena
Donna Converse, MIT X Consortium

NAME

xcalendar - calendar with a notebook for X11

SYNTAX

xcalendar [month [year]]

DESCRIPTION

The *xcalendar* is a simple interactive calendar program with a notebook capability. It is build on the X Toolkit and the Athena Widgets.

If month and year are not provided on the command line they are assumed to be current. To achieve pleasant visual effect you have to load a resource data base to the server. The default database is provided with this program and is also included in this text. Consult the man page for xrdb(1).

INTERACTIONS

Clicking the left mouse button on a day will start a text editor. You can edit and save a text.This text will be associated with the day. You can later on read and edit this text when you open the editor for the same day. The text is saved in a file in the directory ~/Calendar. The editor lets you also clear an entry associated with a particular day.

You can highlight all entries in a month by invoking the function ShowEntries. By default this function is called when the left mouse button is pressed in the title window (where day, month and a year are displayed). Pressing again the same button will unhighlight the entries.

MISSING FEATURES

Currently, to view another month you have to start another process and specify the month and year on the command line. You can run several xcalendars at the same time. However, it would be nice to be able to scroll or browse through the calendar.

To remove all entries in a particular month you have to use your system's commands. The naming scheme for the files makes it easy : the command "rm ~/Calendar/xc*sep1988 " on UNIX(TM) will remove all entries from september 1988. The facility to do that from the xcalendar should be provided.

One can imagine many other useful features. For example automatic parsing of the current day entry in search for appointments to trigger alarms (reminders) at the approriate time. Well, maybe one day...

RESOURCES

The resource data base lets you alter the visual appearance of the program. You can change fonts, border widths, labels, and other resources used by widgets. One use of this facility is to change names of week days and months.

Here are the names of widgets you can use to set various resources:

XCalendar	- class of the application
xcalendar	- top level form
controls	- control panel
quitButton	- quit button
helpButton	- help button
date	- date label
calendar	- calendar frame
daynumbers	- day numbers frame
1-49	- day number buttons
daynames	- day names frame
MON,TUE,WED,THU,FRI,SAT,SUN	- day name buttons
helpWindow	- help window
dayEditor	- editor popup
editorFrame	- editor frame
editorTitle	- editor title
editor	- editor
editorControls	- control panel
doneButton	- done button
saveButton	- save button
clearEntry	- clear entry button

Application specific resources:

reverseVideoMark - if True the entries are highlighted in reverse video; default True for black and white, and False for color displays;

setMarkBackground - if True and reverseVideoMark is False the entries are highlighted by setting background to markBackground ;

markBackground - background color for highlighting entries;

setMarkForeground - analogous to setMarkBackground;

markForeground - foreground color for highlighting entries;

setMarkBackground and *setMarkForeground* can take any combination of values.

january,february,..,december - these resources can be used for changing names of months;

firstDay - an integer between 1-7, indicating the day to start a week with,

```
                    default:  1 (Monday);
markOnStartup  - if True mark the entries upon startup,
                    default: False;
helpFile       - full pathname of the xcalendar.hlp file,
                    default: /usr/lib/X11/xcalendar.hlp;
textBufferSize - maximum size of the text buffer in the day editor,
                    default: 2048;
```

DEFAULT RESOURCE DATA BASE:

```
XCalendar*Font: vtsingle
XCalendar*Background: wheat
XCalendar*BorderWidth:          2
XCalendar*calendar*borderWidth: 0
XCalendar*controls*borderWidth: 0
XCalendar*date*font:            8x13bold
XCalendar*date*borderWidth: 0
XCalendar*daynames*font:                 8x13bold

XCalendar*daynames*Background: PaleGreen

XCalendar*daynames.SUN*Foreground: Red
XCalendar*daynames.SAT*Foreground: DarkGreen

XCalendar*daynumbers.7*Foreground: Red
XCalendar*daynumbers.14*Foreground: Red
XCalendar*daynumbers.21*Foreground: Red
XCalendar*daynumbers.28*Foreground: Red
XCalendar*daynumbers.35*Foreground: Red
XCalendar*daynumbers.42*Foreground: Red


XCalendar*controls*helpButton*Background: DarkGreen
XCalendar*controls*helpButton*Foreground: wheat
XCalendar*controls*quitButton*Background: DarkGreen
XCalendar*controls*quitButton*Foreground: wheat


XCalendar*daynumbers*Foreground: DarkGreen
XCalendar*editorTitle*Background: PaleGreen
XCalendar*editorTitle*Foreground: Red
XCalendar*editorControls*Background: PaleGreen
XCalendar*editorControls*doneButton*Background: Red
XCalendar*editorControls*doneButton*Foreground: wheat
XCalendar*editorControls*saveButton*Background: Red
```

```
XCalendar*editorControls*saveButton*Foreground: wheat
XCalendar*editorControls*clearEntry*Background: DarkGreen
XCalendar*editorControls*clearEntry*Foreground: wheat

XCalendar*dayEditor*geometry: 300x150
XCalendar*dayEditor*editorTitle*font: 8x13bold
XCalendar*dayEditor*editorTitle*Foreground: DarkGreen

XCalendar*helpWindow*geometry: 600x350
XCalendar*helpWindow*editorTitle*font: 8x13bold

XCalendar*doneButton*Label: done
XCalendar*editorTitle*Label: Help
XCalendar*helpButton*Label: help
XCalendar*quitButton*Label: quit
XCalendar*saveButton*Label: save

XCalendar*markBackground: PaleGreen
XCalendar*setMarkBackground: True
```

FILES

$HOME/Calendar/*

SEE ALSO

xrdb(1)

BUGS

Save button handler in the editor cannot detect when a text is pasted. Workaround : type something to the ditor to activate save button.

AUTHOR

Copyright 1988 by Massachusetts Institute of Technology Roman J. Budzianowski, MIT Project Athena

NAME

xclipboard - X clipboard client

SYNOPSIS

xclipboard [*-toolkitoption* ...] [-w] [-nw]

DESCRIPTION

The *xclipboard* program is used to collect and display text selections that are sent to the CLIP_BOARD by other clients. It is typically used to gather together and hold a block of text that has been selected from a variety of different places.

Since *xclipboard* uses a Text Widget to display the contents of the clipboard, text sent to the CLIP_BOARD may be re-selected for use in other applications.

An *xclipboard* window has the following buttons across the top:

quit When this button is pressed, *xclipboard* exits.

erase When this button is pressed, the contents of the text window are erased.

OPTIONS

The *xclipboard* program accepts all of the standard X Toolkit command line options as well as the following:

–w This option indicates that lines of text that are too long to be displayed on one line in the clipboard should wrap around to the following lines.

–nw This option indicates that long lines of text should not wrap around.

SENDING TO CLIPBOARD

Text is copied to the clipboard whenever a client asserts ownership of the **CLIP_BOARD** selection. Examples of event bindings that a user may wish to include in his/her resource configuration file to use the clipboard are:

```
*VT100.Translations: #override \
        Button1 <Btn2Down>:    select-end(CLIPBOARD) \n\
        Button1 <Btn2Up>:      ignore()

*Text.Translations: #override \
        Button1 <Btn2Down>:    extend-end(CLIPBOARD)
```

X DEFAULTS

This program accepts all of the standard X Toolkit resource names and classes as well as:

wordWrap (class **WordWrap**)
This resource specifies whether or not lines of text should wrap around to the following lines. The default is *no*.

SEE ALSO

X(1), xcutsel(1), xterm(1), individual client documentation for how to make a selection and send it to the CLIP_BOARD.

BUGS

The erase button is not yet implemented.

It would be nice to have a way of specifying the file in which the clipboard contents are saved.

FILES

/usr/lib/X11/app-defaults/XClipboard - specifies required resources

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Ralph R. Swick, DEC/MIT Project Athena

NAME

xclock - analog / digital clock for X

SYNOPSIS

xclock [-*toolkitoption* ...] [-option ...]

DESCRIPTION

The *xclock* program displays the time in analog or digital form. The time is continuously updated at a frequency which may be specified by the user. This program is nothing more than a wrapper around the Athena Clock widget.

OPTIONS

Xclock accepts all of the standard X Toolkit command line options along with the additional options listed below:

–help This option indicates that a brief summary of the allowed options should be printed on the standard error.

–analog This option indicates that a conventional 12 hour clock face with tick marks and hands should be used. This is the default.

–digital This option indicates that a 24 hour digital clock should be used.

–chime This option indicates that the clock should chime once on the half hour and twice on the hour.

–hd *color*
This option specifies the color of the hands on an analog clock. The default is *black.*

–hl *color*
This option specifies the color of the edges of the hands on an analog clock, and is only useful on color displays. The default is *black.*

–update *seconds*
This option specifies the frequency in seconds at which *xclock* should update its display. If the clock is obscured and then exposed, it will be updated immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

–padding *number*
This option specifies the width in pixels of the padding between the window border and clock text or picture. The default is 10 on a digital clock and 8 on an analog clock.

The following standard X Toolkit command line arguments are commonly used with *xclock:*

–bg *color*
: This option specifies the color to use for the background of the window. The default is *white*.

–bd *color*
: This option specifies the color to use for the border of the window. The default is *black*.

–bw *number*
: This option specifies the width in pixels of the border surrounding the window.

–fg *color*
: This option specifies the color to use for displaying text. The default is *black*.

–fn *font* This option specifies the font to be used for displaying normal text. The default is *6x10*.

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry*
: This option specifies the prefered size and position of the clock window.

–display *host:display*
: This option specifies the X server to contact.

–xrm *resourcestring*
: This option specifies a resource string to be used.

X DEFAULTS

This program uses the *Clock* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
: Specifies the width of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

height (class **Height**)
: Specifies the height of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

update (class **Interval**)
: Specifies the frequency in seconds at which the time should be redisplayed.

foreground (class **Foreground**)
: Specifies the color for the tic marks. The default is *black* since the core default for background is *white*.

hands (class **Foreground**)
: Specifies the color of the insides of the clock's hands.

highlight (class **Foreground**)
: Specifies the color used to highlight the clock's hands.

analog (class **Boolean**)
: Specifies whether or not an analog clock should be used instead of a digital one. The default is True.

chime (class **Boolean**)
: Specifies whether or not a bell should be rung on the hour and half hour.

padding (class **Margin**)
: Specifies the amount of internal padding in pixels to be used. The default is 8.

font (class **Font**)
: Specifies the font to be used for the digital clock. Note that variable width fonts currently will not always display correctly.

reverseVideo (class **ReverseVideo**)
: Specifies that the foreground and background colors should be reversed.

SEE ALSO

X(1), xrdb(1), time(3C), Athena Clock widget

BUGS

Xclock believes the system clock.

When in digital mode, the string should be centered automatically.

When specifying a time offset, the grammar requires an hours field but if only minutes are given they will be quietly ignored. A negative offset of less than 1 hour is treated as a positive offset.

Digital clock windows default to the analog clock size.

Border color has to be explicitly specified when reverse video is used.

COPYRIGHT

AUTHORS

Tony Della Fera (MIT-Athena, DEC)
Dave Mankins (MIT-Athena, BBN)
Ed Moy (UC Berkeley)

NAME

MIT X Consortium

SYNOPSIS

X11R4 is brought to you by the MIT X Consortium.

DESCRIPTION

X Window System research began in the summer of 1984 at MIT as an informal joint undertaking between the Laboratory for Computer Science and Project Athena. Since that time, researchers and engineers at numerous other universities and companies have been involved in the design and implementation.

The MIT X Consortium was formed in January of 1988 to further the development of the X Window System. It is part of the Laboratory for Computer Science at MIT, and has as its major goal the promotion of cooperation within the computer industry in the creation of standard software interfaces at all layers in the X Window System environment. MIT's role is to provide the vendor-neutral architectural and administrative leadership required to make this work. The Consortium is financially self-supporting, with membership open to any organization. There are two categories of membership, Member (for large organizations) and Affiliate (for smaller organizations). A list of members as of the public release of X11R4 is given below.

The Director of the X Consortium acts as the ultimate architect for all X specifications and software. The activities of the Consortium are overseen by an MIT Steering Committee, which helps set policy and provides strategic guidance and review of the Consortium's activities. An Advisory Committee made up of member representatives meets regularly to review the Consortium's plans, assess its progress, and suggest future directions.

Most of the Consortium's activities take place via electronic mail, with meetings when required. As designs and specifications take shape, interest groups are formed from experts in the participating organizations. Typically a small multi-organization architecture team leads the design, with others acting as close observers and reviewers. Once a complete specification is produced, it may be submitted for formal technical review by the Consortium as a proposed standard. The standards process includes public review (outside the Consortium) and a demonstration of proof of concept, typically established with a public, portable implementation of the specification, (although other methods are possible on a case by case basis).

The MIT staff of the Consortium also maintains a software and documentation collection, containing both Consortium-developed and user-contributed materials, and endeavors to make periodic distributions of this collection available to the public without license and for a minimal fee. X11R4 is the fourth such distribution. The Consortium also sponsors an annual conference, open to the public, to promote the exchange of technical information about X.

STAFF

The Director of the X Consortium is Bob Scheifler, rws@expo.lcs.mit.edu. The MIT staff members are as follows.

Donna Converse, converse@expo.lcs.mit.edu
Jim Fulton, jim@expo.lcs.mit.edu
Michelle Leger, michelle@expo.lcs.mit.edu
Keith Packard, keith@expo.lcs.mit.edu
Chris Peterson, kit@expo.lcs.mit.edu

Ralph Swick, swick@athena.mit.edu, an employee of Digital Equipment Corporation working at MIT Project Athena, also works directly with the Consortium staff.

POSTAL ADDRESS

The MIT X Consortium can be reached by writing to

Bob Scheifler
MIT X Consortium
Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

MEMBERS

Apple Computer, Inc.
American Telephone & Telegraph Company
Bull S.A.
Control Data Corporation
Cray Research, Inc.
Data General Corporation
Digital Equipment Corporation
Eastman Kodak Company
Fujitsu Limited
Hewlett-Packard Company
International Business Machines Corporation
Mitsubishi Electric Corporation
Motorola, Inc.
NEC Corporation
NCR Corporation
Nippon Telegraph and Telephone Corporation

OMRON Corporation
Prime Computer, Inc.
Rich, Inc.
Sequent Computer Systems Inc.
Siemens AG
Silicon Graphics, Inc.
Sony Corporation
Sun Microsystems, Inc.
Tektronix, Inc.
Texas Instruments, Inc.
Unisys Corporation
Wang Laboratories, Inc.
Xerox Corporation

AFFILIATES

ACER America Corporation
Adobe Systems
Advanced Graphics Engineering
Carnegie Mellon University
CETIA - Compagnie Europeene des Techniques de l'Ingenierie Assistee
Codonics, Inc.
Evans & Sutherland
GfxBase
INESC - Instituto de Engenharia de Sistemas e Computadores
Integrated Solutions
Interactive Systems Corporation
Interactive Development Environments
Integrated Computer Solutions, Inc.
Key Systems Engineering Corp.
Jupiter Systems
University of Kent at Canterbury
Landmark Graphics Corporation
Locus Computing
University of Lowell
Matrox International
Megatek Corporation
MIPS Computer Systems
MITRE Corporation
Network Computing Devices
Nova Graphics International
Open Software Foundation
O'Reilly & Associates, Inc.
PCS Computer Systeme GmbH

Ramtek Corporation
Samsung Software America
SGIP - Societe de Gestion et d'Informatique Publicis
Software Productivity Consortium
Solbourne Computer Inc.
Spectragraphics Corporation
Stanford University
Stardent Computer
Tatung Science and Technology
UNICAD, Inc.
Visual Technology Inc.
X/Open Company Ltd.

NAME

xcutsel - interchange between cut buffer and selection

SYNOPSIS

xcutsel [*-toolkitoption* ...] [-selection *selection*] [-cutbuffer *number*]

DESCRIPTION

The *xcutsel* program is used to copy the current selection into a cut buffer and to make a selection that contains the current contents of the cut buffer. It acts as a bridge between applications that don't support selections and those that do.

By default, *xcutsel* will use the selection named PRIMARY and the cut buffer CUT_BUFFER0. Either or both of these can be overridden by command line arguments or by resources.

An *xcutsel* window has the following buttons:

quit When this button is pressed, *xcutsel* exits. Any selections held by *xcutsel* are automatically released.

copy PRIMARY to 0
When this button is pressed, *xcutsel* copies the current selection into the cut buffer.

copy 0 to PRIMARY
When this button is pressed, *xcutsel* converts the current contents of the cut buffer into the selection.

The button labels reflect the selection and cutbuffer selected by command line options or through the resource database.

When the ``copy 0 to PRIMARY'' button is activated, the button will remain inverted as long as *xcutsel* remains the owner of the selection. This serves to remind you which client owns the current selection. Note that the value of the selection remains constant; if the cutbuffer is changed, you must again activate the copy button to retrieve the new value when desired.

OPTIONS

Xcutsel accepts all of the standard X Toolkit command line options as well as the following:

–selection *name*
This option specifies the name of the selection to use. The default is PRIMARY. The only supported abbreviations for this option are ``-select'', ``-sel'' and ``-s'', as the standard toolkit option ``-selectionTimeout'' has a similar name.

−cutbuffer *number*
: This option specifies the cut buffer to use. The default is cut buffer 0.

X DEFAULTS

This program accepts all of the standard X Toolkit resource names and classes as well as:

selection (class **Selection**)
: This resource specifies the name of the selection to use. The default is PRIMARY.

cutBuffer (class **CutBuffer**)
: This resource specifies the number of the cut buffer to use. The default is 0.

WIDGET NAMES

The following instance names may be used when user configuration of the labels in them is desired:

sel-cut (class **Command**)
: This is the "copy SELECTION to BUFFER" button.

cut-sel (class **Command**)
: This is the "copy BUFFER to SELECTION" button.

quit (class **Command**)
: This is the "quit" button.

SEE ALSO

X(1), xclipboard(1), xterm(1), text widget documentation, individual client documentation for how to make a selection.

BUGS

There is no way to change the name of the selection or the number of the cut buffer while the program is running.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Ralph R. Swick, DEC/MIT Project Athena

NAME

xditview — display ditroff DVI files

SYNOPSIS

xditview [-*toolkitoption* ...] [-option ...]

DESCRIPTION

The *xditview* program displays ditroff output on an X display. It uses special font metrics which match the font set distributed with X11vR3, so it does not require access to the server machine for font loading.

OPTIONS

Xditview accepts all of the standard X Toolkit command line options along with the additional options listed below:

–help This option indicates that a brief summary of the allowed options should be printed.

–page This option specifies the page number of the document to be displayed.

–backingStore *backing-store-type*
Redisplay of the DVI window can take upto a second or so, this option causes the server to save the window contents so that when it is scrolled around the viewport, the window is painted from contents saved in backing store. *backing-store-type* can be one of **Always, WhenMapped** or **NotUseful.**

The following standard X Toolkit command line arguments are commonly used with *xditview:*

–bg *color*
This option specifies the color to use for the background of the window. The default is *white.*

–bd *color*
This option specifies the color to use for the border of the window. The default is *black.*

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color*
This option specifies the color to use for displaying text. The default is *black.*

–fn *font* This option specifies the font to be used for displaying widget text. The default is *fixed.*

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry*
This option specifies the prefered size and position of the window.

–display *host:display*
This option specifies the X server to contact.

–xrm *resourcestring*
This option specifies a resource string to be used.

X DEFAULTS

This program uses the *Dvi* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the window.

height (class **Height**)
Specifies the height of the window.

foreground (class **Foreground**)
Specifies the default foreground color.

font (class **Font**)
Specifies the font to be used for error messages.

USING XDITVIEW WITH DITROFF

To build a DVI file suitable for use with xditview, use the device description in devX75:

```
$ cd devX75
$ makedev DESC
$ mkdir /usr/lib/font/devX75
$ cp *.out /usr/lib/font/devX75
$ ditroff -TX75 ditroff-input | xditview
```

SEE ALSO

X(1), xrdb(1), ditroff(1)

BUGS

Xditview can be easily confused by attempting to display a DVI file constructed for the wrong device.

ORIGIN

Portions of this program originated in xtroff which was derived from suntroff.

COPYRIGHT

AUTHORS

Keith Packard (MIT X Consortium)
Richard L. Hyde (Purdue)
David Slattengren (Berkeley)
Malcom Slaney (Schlumberger Palo Alto Research)
Mark Moraes (University of Toronto)

NAME

xdm – X Display Manager

SYNOPSIS

xdm [-config *configuration_file*] [-daemon] [-debug *debug_level*] [-error *error_log_file*] [-nodaemon] [-resources *resource_file*] [-server *server_entry*] [-session *session_program*] [-xrm *resource_specification*]

DESCRIPTION

Xdm manages a collection of X displays, both local and possibly remote — the emergence of X terminals guided the design of several parts of this system, along with the development of the X Consortium standard XDMCP, the *X Display Manager Control Protocol*). It is designed to provide services similar to that provided by init, getty and login on character terminals: prompting for login/password, authenticating the user and running a ''session''.

A ''session'' is defined by the lifetime of a particular process; in the traditional character-based terminal world, it is the user's login shell process. In the *xdm* context, it is an arbitrary session manager. This is because in a windowing environment, a user's login shell process would not necessarily have any terminal-like interface with which to connect.

Until real session managers become widely available, the typical *xdm* substitute would be either a window manager with an exit option, or a terminal emulator running a shell - with the condition that the lifetime of the terminal emulator is the lifetime of the shell process that it is running - thus degenerating the X session to an emulation of the character-based terminal session.

When the session is terminated, *xdm* resets the X server and (optionally) restarts the whole process.

Because *xdm* provides the first interface that users will see, it is designed to be simple to use and easy to customize to the needs of a particular site. *Xdm* has many options, most of which have reasonable defaults. Browse through the various sections, picking and choosing the things you want to change. Pay particular attention to the Xsession section, which will describe how to set up the style of session desired.

OPTIONS

First, note that all of these options, except -config, specify values which can also be specified in the configuration file as resources.

-config *configuration_file*
: Specifies a resource file which specifies the remaining configuration parameters. If no file is specified and the file */usr/lib/X11/xdm/xdm-config* exists, *xdm* will use it.

-daemon
Specifies "true" as the value for the **DisplayManager.daemonMode** resource. This makes *xdm* close all file descriptors, disassociate the controlling terminal and put itself in the background when it first starts up (just like the host of other daemons). It is the default behavior.

-debug *debug_level*
Specifies the numeric value for the **DisplayManager.debugLevel** resource. A non-zero value causes *xdm* to print piles of debugging statements to the terminal; it also disables the **DisplayManager.daemonMode** resource, forcing *xdm* to run synchronously. To interpret these debugging messages, a copy of the source code for xdm is almost a necessity. No attempt has been made to rationalize or standardize the output.

-error *error_log_file*
Specifies the value for the **DisplayManager.errorLogFile** resource. This file contains errors from *xdm* as well as anything written to stderr by the various scripts and programs run during the progress of the session.

-nodaemon
Specifies "false" as the value for the **DisplayManager.daemonMode** resource.

-resources *resource_file*
Specifies the value for the **DisplayManager*resources** resource. This file is loaded using *xrdb (1)* to specify configuration parameters for the authentication widget.

-server *server_entry*
Specifies the value for the **DisplayManager.servers** resource. See the section below which describes this resource in depth.

-udpPort *port_number*
Specifies the value for the **DisplayManager.requestPort** resource. This sets the port-number which XDM will monitor for XDMCP requests. As XDMCP uses the registered well-known udp port 177, this resource should probably not be changed except for debugging.

-session *session_program*
Specifies the value for the **DisplayManager*session** resource. This indicates the program to run when the user has logged in as the session.

-xrm *resource_specification*

This allows an arbitrary resource to be specified, just as most toolkit applications.

RESOURCES

At many stages the actions of *xdm* can be controlled through the use of the configuration file, which is in the familiar X resource format. See Jim Fulton's article on resource files (*doc/tutorials/resources.txt*) for a description of the format. Some resources modify the behavior of *xdm* on all displays, while others modify its behavior on one single display. Where actions relate to a specific display, the display name is inserted into the resource name between "DisplayManager" and the final resource name segment. For example, **DisplayManager.expo_0.startup** is the name of the resource which defines the startup shell file on the "expo:0" display. Because the resource manager uses colons to separate the name of the resource from its value and dots to separate resource name parts, *xdm* substitutes underscores for the dots and colons when generating the resource name.

DisplayManager.servers

This resource either specifies a file name full of server entries, one per line (if the value starts with a slash), or a single server entry. Each entry indicates a displays which should constantly be managed and which is not using XDMCP. Each entry consists of at least three parts: a display name, a display class, a display type, and (for local servers) a command line to start the server. A typical entry for local display number 0 would be:

```
:0 Digital-QV local /usr/bin/X11/X :0
```

The display types are:

local	a local display, i.e. one which has a server program to run
foreign	a remote display, i.e. one which has no server program to run

The display name must be something that can be passed in the **-display** option to any X program. This string is used in the display-specific resources to specify the particular display, so be careful to match the names (e.g. use ":0 local /usr/bin/X11/X :0" instead of "localhost:0 local /usr/bin/X11/X :0" if your other resources are specified as "DisplayManager._0.session"). The display class portion is also used in the display-specific resources, as the class portion of the resource. This is useful if you have a large collection of similar displays (like a corral of X terminals) and would like to set resources for groups of them. When using XDMCP, the display is required to specify the display class, so perhaps your X terminal documentation describes a reasonably standard display class string for your device.

DisplayManager.requestPort
This indicates the UDP port number which *xdm* uses to listen for incoming XDMCP requests. Unless you need to debug the system, leave this with it's default value of 177.

DisplayManager.errorLogFile
Error output is normally directed at the system console. To redirect it simply set this resource to any file name. A method to send these messages to syslog should be developed for systems which support it; however the wide variety of "standard" interfaces precludes any system-independent implementation. This file also contains any output directed to stderr by *Xstartup, Xsession and Xreset*, so it will contain descriptions of problems in those scripts as well.

DisplayManager.debugLevel
A non-zero value specified for this integer resource will enable reams of debugging information to be printed. It also disables daemon mode which would redirect the information into the bit-bucket. Specifying a non-zero debug level also allows non-root users to run *xdm* which would normally not be useful.

DisplayManager.daemonMode
Normally, *xdm* attempts to make itself into an unassociated daemon process. This is accomplished by forking and leaving the parent process to exit, then closing file descriptors and mangling the controlling terminal. When attempting to debug *xdm* , this is quite bothersome. Setting this resource to "false" will disable this feature.

DisplayManager.pidFile
The filename specified will be created to contain an ascii representation of the process-id of the main xdm process. This is quite useful when reinitializing the system. *Xdm* also uses file locking to attempt to eliminate multiple daemons running on the same machine, which would cause quite a bit of havoc.

DisplayManager.lockPidFile
This is the resource which controls whether *xdm* uses file locking to keep multiple xdms from running amok. On SYSV, this uses the lockf library call, while on BSD it uses flock. The default value is "true".

DisplayManager.remoteAuthDir
This is a directory name which *xdm* uses to temporarily store authorization files for displays using XDMCP. The default value is /usr/lib/X11/xdm.

DisplayManager.autoRescan"
This boolean controls whether *xdm* rescans the configuration file and servers file after a session terminates and the files have changed. By default it is "true". You can force *xdm* to reread these files by sending a SIGHUP to the main process.

DisplayManager.removeDomainname
When computing the display name for XDMCP clients, the resolver will typically create a fully qualified host name for the terminal. As this is sometimes confusing, *xdm* will remove the domain name portion of the host name if it is the same as the domain name for the local host when this variable is set. By default the value is "true".

DisplayManager.keyFile
XDM-AUTHENTICATION-1 style XDMCP authentication requires that a private key be shared between *xdm* and the terminal. This resource specifies the file containing those values. Each entry in the file consists of a display name and the shared key. By default, *xdm* does not include support for XDM-AUTHENTICATION-1 as it requires DES which is not generally distributable.

DisplayManager.DISPLAY.resources
This resource specifies the name of the file to be loaded by *xrdb (1)* as the resource data-base onto the root window of screen 0 of the display. This resource data base is loaded just before the authentication procedure is started, so it can control the appearance of the "login" window. See the section below on the authentication widget which describes the various resources which are appropriate to place in this file. There is no default value for this resource, but the conventional name is /usr/lib/X11/xdm/Xresources.

DisplayManager.DISPLAY.xrdb
Specifies the program used to load the resources. By default, *xdm* uses */usr/bin/X11/xrdb.*

DisplayManager.DISPLAY.cpp
This specifies the name of the C preprocessor which is used by xrdb.

DisplayManager.DISPLAY.startup

This specifies a program which is run (as root) after the authentication process succeeds. By default, no program is run. The conventional name for a file used here is *Xstartup*. See the **Xstartup** section below.

DisplayManager.DISPLAY.session

This specifies the session to be executed (not running as root). By default, */usr/bin/X11/xterm* is run. The conventional name is *Xsession*. See the **Xsession** session below.

DisplayManager.DISPLAY.reset

This specifies a program which is run (as root) after the session terminates. Again, by default no program is run. The conventional name is *Xreset*. See the **Xreset** section further on in this document.

DisplayManager.DISPLAY.openDelay

DisplayManager.DISPLAY.openRepeat

DisplayManager.DISPLAY.openTimeout

DisplayManager.DISPLAY.startAttempts

These numeric resources control the behavior of *xdm* when attempting to open intransigent servers. **openDelay** is the length of the pause (in seconds) between successive attempts, **openRepeat** is the number of attempts to make, **openTimeout** is the amount of time to wait while actually attempting the open (i.e. the maximum time spent in the *connect (2)* syscall) and **startAttempts** is the number of times this entire process is done before giving up on the server. After **openRepeat** attempts have been made, or if **openTimeout** seconds elapse in any particular attempt, *xdm* terminates and restarts the server, attempting to connect again, this process is repeated **startAttempts** time, at which point the display is declared dead and disabled. Although this behavior may seem arbitrary, it has been empirically developed and works quite well on most systems. The default values are 5 for **openDelay**, 5 for **openRepeat**, 30 for **openTimeout** and 4 for **startAttempts**.

DisplayManager.DISPLAY.pingInterval

DisplayManager.DISPLAY.pingTimeout

To discover when remote displays disappear, *xdm* occasionally "pings" them, using an X connection and sending XSync requests. **pingInterval** specifies the time (in minutes) between each ping attempt, **pingTimeout** specifies the maximum amount of time (in minutes) to wait for the terminal to respond to the request. If the terminal does not respond, the session is declared dead and terminated. By default, both are set to 5 minutes. *xdm* will not ping local displays. Although it would seem harmless, it is unpleasant when the workstation session is terminated as a result of the server hanging for NFS service and not responding to the ping.

DisplayManager.DISPLAY.terminateServer

This boolean resource specifies whether the X server should be terminated when a session terminates (instead of resetting it). This option can be used when the server tends to grow without bound over time in order to limit the amount of time the server is run. The default value is "FALSE".

DisplayManager.DISPLAY.userPath

Xdm sets the PATH environment variable for the session to this value. It should be a colon separated list of directories, see *sh(1)* for a full description. The default value can be specified in the X system configuration file with DefUserPath, frequently it is set to ":/bin:/usr/bin:/usr/bin/X11:/usr/ucb".

DisplayManager.DISPLAY.systemPath

Xdm sets the PATH environment variable for the startup and reset scripts to the value of this resource. The default for this resource is specified with the DefaultSystemPath entry in the system configuration file, but it is frequently "/etc:/bin:/usr/bin:/usr/bin/X11:/usr/ucb". Note the conspicuous absence of "." from this entry. This is a good practise to follow for root; it avoids many common trojan horse system penetration schemes.

DisplayManager.DISPLAY.systemShell

Xdm sets the SHELL environment variable for the startup and reset scripts to the value of this resource. By default, it is "/bin/sh".

DisplayManager.DISPLAY.failsafeClient

If the default session fails to execute, *xdm* will fall back to this program. This program is executed with no arguments, but executes using the same environment variables as the session would have had (see the section "Xsession" below). By default, */usr/bin/X11/xterm* is used.

DisplayManager.DISPLAY.grabServer

DisplayManager.DISPLAY.grabTimeout

To eliminate obvious security shortcomings in the X protocol, *xdm* grabs the server and keyboard while reading the name/password. The grabServer resource specifies if the server should be held for the duration of the name/password reading, when FALSE, the server is ungrabbed after the keyboard grab succeeds, otherwise the server is grabbed until just before the session begins. The **grabTimeout** resource specifies the maximum time *xdm* will wait for the grab to succeed. The grab may fail if some other client has the server grabbed, or possibly if the network latencies are very high. This resource has a default value of 3 seconds; you should be cautious when raising it as a user can be spoofed by a look-alike window on the display. If the grab fails, *xdm* kills and restarts the server (if possible) and session.

DisplayManager.DISPLAY.authorize

DisplayManager.DISPLAY.authName

authorize is a boolean resource which controls whether *xdm* generates and uses authorization for the server connections. If authorization is used, **authName** specifies the type to use. Currently, xdm supports only MIT-MAGIC-COOKIE-1 authorization, XDM-AUTHORIZATION-1 could be supported as well, but DES is not generally distributable. XDMCP connections specify which authorization types are supported dynamically, so **authName** is ignored in this case. When **authorize** is set for a display and authorization is not available, the user is informed by having a different message displayed in the login widget. By default, **authorize** is "true"; **authName** is "MIT-MAGIC-COOKIE-1".

DisplayManager.DISPLAY.authFile"

This file is used to communicate the authorization data from xdm to the server, using the *-auth* server command line option. It should be kept in a directory which is not world-writable as it could easily be removed, disabling the authorization mechanism in the server.

DisplayManager.DISPLAY.resetForAuth

The original implementation of authorization in the sample server reread the authorization file at server reset time, instead of when checking the initial connection. As *xdm* generates the authorization information just before connecting to the display, an old server would not get up-to-date authorization information. This resource causes *xdm* to send SIGHUP to the server after setting up the file, causing an additional server reset to occur, during which time the new authorization information will be read

DisplayManager.DISPLAY.userAuthDir

When *xdm* is unable to write to the usual user authorization file ($HOME/.Xauthority), it creates a unique file name in this directory and points the environment variable XAUTHORITY at the created file. By default it uses "/tmp".

CONTROLLING THE SERVER

Xdm controls local servers using POSIX signals. SIGHUP is expected to reset the server, closing all client connections and performing other clean up duties. SIGTERM is expected to terminate the server. If these signals do not perform the expected actions, *xdm* will not perform properly.

To control remote servers not using XDMCP, *xdm* searches the window hierarchy on the display and uses the protocol request KillClient in an attempt to clean up the terminal for the next session. This may not actually kill all of the clients, as only those which have created windows will be noticed. XDMCP provides a more sure mechanism; when xdm closes it's initial connection, the session is over and the terminal is required to close all other connections.

CONTROLLING XDM

Xdm responds to two signals: SIGHUP and SIGTERM. When sent a SIGHUP, *xdm* rereads the configuration file and the 1file specified by the **DisplayManager.servers** resource and notices if entries have been added or removed. If a new entry has been added, *xdm* starts a session on the associated display. Entries which have been removed are disabled immediately, meaning that any session in progress will be terminated without notice, and no new session will be started.

When sent a SIGTERM, *xdm* terminates all sessions in progress and exits. This can be used when shutting down the system.

Xdm attempts to mark the various sub-processes for ps(1) by editing the command line argument list in place. Because xdm can't allocate additional space for this task, it is useful to start xdm with a reasonably long command line (15 to 20 characters should be enough). Each process which is servicing a display is marked "-<Display-Name>".

AUTHENTICATION WIDGET

The authentication widget is an application which reads a name/password pair from the keyboard. As this is a toolkit client, nearly every imaginable parameter can be controlled with a resource. Resources for this widget should be put into the file named by **DisplayManager.DISPLAY.resources**. All of these have reasonable default values, so it is not necessary to specify any of them.

xlogin.Login.width, xlogin.Login.height, xlogin.Login.x, xlogin.Login.y
The geometry of the login widget is normally computed automatically. If you wish to position it elsewhere, specify each of these resources.

xlogin.Login.foreground
The color used to display the typed-in user name.

xlogin.Login.font
The font used to display the typed-in user name.

xlogin.Login.greeting
A string which identifies this window. The default is "Welcome to the X Window System".

xlogin.Login.unsecureGreeting
When X authorization is requested in the configuration file for this display and none is in use, this greeting replaces the standard greeting. It's default value is "This is an unsecure session"

xlogin.Login.greetFont
The font used to display the greeting.

xlogin.Login.greetColor
The color used to display the greeting.

xlogin.Login.namePrompt
The string displayed to prompt for a user name. *Xrdb* strips trailing white space from resource values, so to add spaces at the end of the prompt (usually a nice thing), add spaces escaped with backslashes. The default is "Login: "

xlogin.Login.passwdPrompt
The string displayed to prompt for a password. The default is "Password: ".

xlogin.Login.promptFont
The font used to display both prompts.

xlogin.Login.promptColor
The color used to display both prompts.

xlogin.Login.fail
A message which is displayed when the authentication fails. The default is "Login Failed, please try again".

xlogin.Login.failFont
The font used to display the failure message.

xlogin.Login.failColor
The color used to display the failure message.

xlogin.Login.failTimeout
The time (in seconds) that the fail message is displayed. The default is 30 seconds.

xlogin.Login.translations

This specifies the translations used for the login widget. Refer to the X Toolkit documentation for a complete discussion on translations. The default translation table is:

```
Ctrl<Key>H:          delete-previous-character() \n\
Ctrl<Key>D:          delete-character() \n\
Ctrl<Key>B:          move-backward-character() \n\
Ctrl<Key>F:          move-forward-character() \n\
Ctrl<Key>A:          move-to-begining() \n\
Ctrl<Key>E:          move-to-end() \n\
Ctrl<Key>K:          erase-to-end-of-line() \n\
Ctrl<Key>U:          erase-line() \n\
Ctrl<Key>X:          erase-line() \n\
Ctrl<Key>C:          restart-session() \n\
Ctrl<Key>\\:         abort-session() \n\
<Key>BackSpace:      delete-previous-character() \n\
<Key>Delete:         delete-previous-character() \n\
<Key>Return:         finish-field() \n\
<Key>:               insert-char() \
```

The actions which are supported by the widget are:

delete-previous-character
: Erases the character before the cursor.

delete-character
: Erases the character after the cursor.

move-backward-character
: Moves the cursor backward.

move-forward-character
: Moves the cursor forward.

move-to-begining
: (Apologies about the spelling error.) Moves the cursor to the beginning of the editable text.

move-to-end
: Moves the cursor to the end of the editable text.

erase-to-end-of-line
: Erases all text after the cursor.

erase-line
: Erases the entire text.

finish-field
: If the cursor is in the name field, proceeds to the password field; if the cursor is in the password field, check the current name/password pair. If the name/password pair are valid, *xdm* starts the session. Otherwise the failure message is displayed and the user is prompted to try again.

abort-session
: Terminates and restarts the server.

abort-display
: Terminates the server, disabling it. This is a rash action and is not accessible in the default configuration. It can be used to stop *xdm* when shutting the system down, or when using xdmshell.

restart-session
: Resets the X server and starts a new session. This can be used when the resources have been changed and you want to test them, or when the screen has been overwritten with system messages.

insert-char
: Inserts the character typed.

set-session-argument
: Specifies a single word argument which is passed to the session at startup. See the sections on Xsession and Typical usage.

allow-all-access
: Disables access control in the server, this can be used when the .Xauthority file cannot be created by xdm. Be very careful using this, it might be better to disconnect the machine from the network before doing this.

The Xstartup file

This file is typically a shell script. It is run as "root" and should be very careful about security. This is the place to put commands which make fake entries in /etc/utmp, mount users' home directories from file servers, display the message of the day, or abort the session if logins are not allowed. Various environment variables are set for the use of this script:

DISPLAY	is set to the associated display name
HOME	is set to the home directory of the user
USER	is set to the user name
PATH	is set to the value of DisplayManager.DISPLAY.systemPath

SHELL	is set to the value of **DisplayManager.DISPLAY.systemShell**
XAUTHORITY	may be set to an authority file

No arguments of any kind are passed to the script. *Xdm* waits until this script exits before starting the user session. If the exit value of this script is non-zero, *xdm* discontinues the session immediately and starts another authentication cycle.

The Xsession program

This is the command which is run as the user's session. It is run with the permissions of the authorized user, and has several environment variables specified:

DISPLAY	is set to the associated display name
HOME	is set to the home directory of the user
USER	is set to the user name
PATH	is set to the value of **DisplayManager.DISPLAY.userPath**
SHELL	is set to the user's default shell (from /etc/passwd)
XAUTHORITY	may be set to a non-standard authority file

At most installations, *Xsession* should look in $HOME for a file *.xsession* which would contain commands that each user would like to use as a session. This would replace the system default session. *Xsession* should also implement the system default session if no user-specified session exists. See the section **Typical Usage** below.

An argument may be passed to this program from the authentication widget using the 'set-session-argument' action. This can be used to select different styles of session. One very good use of this feature is to allow the user to escape from the ordinary session when it fails. This would allow users to repair their own *.xsession* if it fails, without requiring administrative intervention. The section on typical usage demonstrates this feature.

The Xreset file

Symmetrical with *Xstartup*, this script is run after the user session has terminated. Run as root, it should probably contain commands that undo the effects of commands in *Xstartup*, removing fake entries from */etc/utmp* or unmounting directories from file servers. The collection of environment variables that were passed to *Xstartup* are also given to *Xreset*.

Typical Usage

Actually, *xdm* is designed to operate in such a wide variety of environments that "typical" is probably a misnomer. However, this section will focus on making *xdm* a superior solution to traditional means of starting X from /etc/ttys or manually.

First off, the *xdm* configuration file should be set up. A good thing to do is to make a directory (*/usr/lib/X11/xdm* comes immediately to mind) which will contain all of the relevant files. Here is a reasonable configuration file, which could be named *xdm-config* :

```
DisplayManager.servers:          /usr/lib/X11/xdm/Xservers
DisplayManager.errorLogFile:     /usr/lib/X11/xdm/xdm-errors
DisplayManager.pidFile:          /usr/lib/X11/xdm/xdm-pid
DisplayManager*resources:        /usr/lib/X11/xdm/Xresources
DisplayManager*session:          /usr/lib/X11/xdm/Xsession
DisplayManager._0.authorize:     true
DisplayManager*authorize:        false
```

As you can see, this file simply contains references to other files. Note that some of the resources are specified with "*" separating the components. These resources can be made unique for each different display, by replacing the "* " with the display-name, but normally this is not very useful. See the **Resources** section for a complete discussion.

The first file */usr/lib/X11/xdm/Xservers* contains the list of displays to manage. Most workstations have only one display, numbered 0, so the file will look like this:

```
          :0 Local local /usr/bin/X11/X :0
```

This will keep */usr/bin/X11/X* running on this display and manage a continuous cycle of sessions.

The file */usr/lib/X11/xdm/xdm-errors* will contain error messages from *xdm* and anything output to stderr by *Xstartup, Xsession or Xreset.* When you have trouble getting *xdm* working, check this file to see if *xdm* has any clues to the trouble.

The next configuration entry, */usr/lib/X11/xdm/Xresources*, is loaded onto the display as a resource database using *xrdb (1).* As the authentication widget reads this database before starting up, it usually contains parameters for that widget:

```
xlogin*login.translations: #override\
   <Key>F1: set-session-argument(failsafe) finish-field()\n\
   <Key>Return: set-session-argument() finish-field()
xlogin*borderWidth: 3
#ifdef COLOR
xlogin*greetColor: #f63
xlogin*failColor: red
xlogin*Foreground: black
```

```
xlogin*Background: #fdc
#else
xlogin*Foreground: black
xlogin*Background: white
#endif
```

The various colors specified here look reasonable on several of the displays we have, but may look awful on other monitors. As X does not currently have any standard color naming scheme, you might need to tune these entries to avoid disgusting results. Please note the translations entry; it specifies a few new translations for the widget which allow users to escape from the default session (and avoid troubles that may occur in it). Note that if #override is not specified, the default translations are removed and replaced by the new value, not a very useful result as some of the default translations are quite useful (like "<Key>: insert-char ()" which responds to normal typing).

The *Xstartup* file used here simply prevents login while the file */etc/nologin* exists. As there is no provision for displaying any messages here (there isn't any core X client which displays files), the user will probably be baffled by this behavior. I don't offer this as a complete example, but simply a demonstration of the available functionality.

Here is a sample *Xstartup* script:

```
#!/bin/sh
#
# Xstartup
#
# This program is run as root after the user is verified
#
if [ -f /etc/nologin ]; then
        exit 1
fi
exit 0
```

The most interesting script is *Xsession*. This version recognizes the special "failsafe" mode, specified in the translations in the *Xresources* file above, to provide an escape from the ordinary session:

```
#!/bin/sh
#
# Xsession
#
# This is the program that is run as the client
# for the display manager.  This example is
# quite friendly as it attempts to run a per-user
# .xsession file instead of forcing a particular
# session layout
#

case $# in
1)
        case $1 in
        failsafe)
                exec xterm -geometry 80x24-0-0 -ls
                ;;
        esac
esac

startup=$HOME/.xsession
resources=$HOME/.Xresources

if [ -f $startup ]; then
        exec $startup
        exec /bin/sh $startup
else
        if [ -f $resources ]; then
                xrdb -load $resources
        fi
        twm &
        exec xterm -geometry 80x24+10+10 -ls
fi
```

No *Xreset* script is necessary, so none is provided.

SOME OTHER POSSIBILITIES

You can also use *xdm* to run a single session at a time, using the 4.3 *init* options or other suitable daemon by specifying the server on the command line:

```
xdm -server ":0 SUN-3/60CG4 local /usr/bin/X :0"
```

Or, you might have a file server and a collection of X terminals. The configuration for this could look identical to the sample above, except the *Xservers* file might look like:

```
extol:0 VISUAL-19 foreign
exalt:0 NCD-19 foreign
explode:0 NCR-TOWERVIEW3000 foreign
```

This would direct *xdm* to manage sessions on all three of these terminals. See the section "Controlling Xdm" above for a description of using signals to enable and disable these terminals in a manner reminiscent of init(8).

One thing that *xdm* isn't very good at doing is coexisting with other window systems. To use multiple window systems on the same hardware, you'll probably be more interested in *xinit* .

SEE ALSO

X(1), xinit(1) and XDMCP

BUGS

COPYRIGHT

AUTHOR

Keith Packard, MIT X Consortium

NAME

xdpr – dump an X window directly to a printer

SYNOPSIS

xdpr [*filename*] [**–display** *host:display*] [**–P***printer*] [**–device** *printer_device*] [option ...]

DESCRIPTION

Xdpr uses the commands *xwd*(1), *xpr*(1), and *lpr*(1) to dump an X window, process it for a particular printer type, and print it out on the printer of your choice. This is the easiest way to get a printout of a window. *Xdpr* by default will print the largest possible representation of the window on the output page.

The options for *xdpr* are the same as those for *xpr*, *xwd*, and *lpr*. The most commonly-used options are described below; see the manual pages for these commands for more detailed descriptions of the many options available.

filename
: Specifies a file containing a window dump (created by *xwd*) to be printed instead of selecting an X window.

-P*printer*
: Specifies a printer to send the output to. If a printer name is not specified here, *xdpr* (really, *lpr*) will send your output to the printer specified by the *PRINTER* environment variable. Be sure that type of the printer matches the type specified with the *–device* option.

-display *host:display*[*.screen*]
: Normally, *xdpr* gets the host and display number to use from the environment variable "DISPLAY". One can, however, specify them explicitly; see *X*(1).

-device *printer-device*
: Specifies the device type of the printer. Available printer devices are "ln03" for the DEC LN03, "pp" for the IBM 3812 PagePrinter, and "ps" for any postscript printer (e.g. DEC LN03R or LPS40). The default is "ln03".

-help This option displays the list of options known to xdpr.

Any other arguments will be passed to the *xwd*(1), *xpr*(1), and *lpr*(1) commands as appropriate for each.

SEE ALSO

xwd(1), xpr(1), lpr(1), xwud(1), X(1)

ENVIRONMENT

DISPLAY - for which display to use by default.
PRINTER - for which printer to use by default.

COPYRIGHT

AUTHOR

Paul Boutin, MIT Project Athena
Michael R. Gretzinger, MIT Project Athena
Jim Gettys, MIT Project Athena

NAME

xdpyinfo - display information utility for X

SYNOPSIS

xdpyinfo [-display *displayname*]

DESCRIPTION

Xdpyinfo is a utility for displaying information about an X server. It is used to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the different types of screens and visuals that are available.

EXAMPLE

The following shows a sample produced by *xdpyinfo* when connected to display that supports an 8 plane Pseudocolor screen as well as a 1 plane (monochrome) screen.

```
name of display:    empire:0.0
version number:    11.0
vendor string:    MIT X Consortium
vendor release number:    3
maximum request size:  16384 longwords (65536 bytes)
motion buffer size:  0
bitmap unit, bit order, padding:    32, MSBFirst, 32
image byte order:    MSBFirst
number of supported pixmap formats:    2
supported pixmap formats:
   depth 1, bits_per_pixel 1, scanline_pad 32
   depth 8, bits_per_pixel 8, scanline_pad 32
keycode range:    minimum 8, maximum 129
default screen number:    0
number of screens:    2

screen #0:
 dimensions:    1152x900 pixels (325x254 millimeters)
 resolution:    90x90 dots per inch
 root window id:    0x8006d
 depth of root window:    1 plane
 number of colormaps:    minimum 1, maximum 1
 default colormap:    0x80065
 default number of colormap cells:    2
 preallocated pixels:    black 1, white 0
 options:    backing-store YES, save-unders YES
 current input event mask:    0x1b8003c
  ButtonPressMask        ButtonReleaseMask      EnterWindowMask
  LeaveWindowMask        SubstructureNotifyMask  SubstructureRedirectMask
```

```
    FocusChangeMask        ColormapChangeMask     OwnerGrabButtonMask
  number of visuals:   1
  default visual id:  0x80064
  visual:
    visual id:   0x80064
    class:   StaticGray
    depth:   1 plane
    size of colormap:   2 entries
    red, green, blue masks:   0x0, 0x0, 0x0
    significant bits in color specification:   1 bits

screen #1:
  dimensions:   1152x900 pixels (325x254 millimeters)
  resolution:   90x90 dots per inch
  root window id:   0x80070
  depth of root window:   8 planes
  number of colormaps:   minimum 1, maximum 1
  default colormap:   0x80067
  default number of colormap cells:   256
  preallocated pixels:   black 1, white 0
  options:   backing-store YES, save-unders YES
  current input event mask:   0x0
  number of visuals:   1
  default visual id:  0x80066
  visual:
    visual id:   0x80066
    class:   PseudoColor
    depth:   8 planes
    size of colormap:   256 entries
    red, green, blue masks:   0x0, 0x0, 0x0
    significant bits in color specification:   8 bits
```

ENVIRONMENT

DISPLAY
To get the default host, display number, and screen.

SEE ALSO

X(1), xwininfo(1), xprop(1), xrdb(1)

BUGS

Due to a bug in the Xlib interface, there is currently no portable way to determine the depths of pixmap images that are supported by the server.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xedit - simple text editor for X

SYNTAX

xedit [*-toolkitoption* ...] [filename]

OPTIONS

Xedit accepts all of the standard X Toolkit command line options (see *X(1)*), plus:

filename	Specifies the file that is to be loaded during start-up. This is the file which will be edited. If a file is not specified, *xedit* lets you load a file or create a new file after it has started up.

DESCRIPTION

Xedit provides a window consisting of the following three areas:

Commands Menu	Lists editing commands (for example, **Undo** or **Search**).
Message Window	Displays *xedit* messages. In addition, this window can be used as a scratch pad.
Edit Window	Displays the text of the file that you are editing or creating.

COMMANDS

Quit	Quits the current editing session. If any changes have not been saved, *xedit* displays a warning message and allows you to save the file.
Save	Stores a copy of the original, unedited file in *file*.BAK. Then, overwrites the original file with the edited contents.
Edit	Allows the text displayed in the Edit window to be edited.
Load	Loads the specified file and displays it in the Edit window.
Undo	Undoes the last edit only.
More	Undoes each edit previous to the last edit, which must first be undone with the **Undo** command.
Jump	Advances the cursor from the beginning of the file to the text line that corresponds to the selected line number.

<<	Searches from the cursor back to the beginning of the file for the string entered in the Search input box. If you do not enter a string in the Search input box, *xedit* automatically copies the last string that you selected from any X application into the Search input box and searches for that string.
Search >>	Searches from the cursor forward to the end of the file for the string entered in the search input box. If you do not enter a string in the Search input box, *xedit* automatically copies the last string that you selected from any X application into the Search input box and searches for that string.
Replace	Replaces the last searched-for string with the string specified in the Replace input box. If no string has been previously searched for, searches from the insert cursor to the end of the file for the next occurrence of the search string and highlights it.
All	Repositions the cursor at the beginning of the file and replaces all occurrences of the search string with the string specified in the Replace input box.

X DEFAULTS

For *xedit*, the available class identifiers are:

ButtonBox
Command
Scrollbar
Text

For *xedit*, the available name identifiers are:

All
Edit
EditWindow
Jump
Load
MessageWindow
More
Quit
Replace

Save
Undo
xedit

For *xedit*, the available resources are:

EnableBackups	Specifies that, when edits made to an existing file are saved, *xedit* is to copy the original version of that file to *file*.BAK before it saves the changes. If the value of this option is specified as off, a backup file is not created.
background	Specifies the background color to be displayed in command buttons. The default is white.
border	Specifies the border color of the *xedit* window.
borderWidth	Specifies the border width, in pixels, of the *xedit* window.
font	Specifies the font displayed in the *xedit* window.
foreground	Specifies the foreground color of the *xedit* window. The default is black.
geometry	Specifies the geometry (window size and screen location) to be used as the default for the *xedit* window. For information about the format of the geometry specification, see *X(1)*.
internalHeight	Specifies the internal horizontal padding (spacing between text and button border) for command buttons.
internalWidth	Specifies the internal vertical padding (spacing between text and button border) for command buttons.

KEY BINDINGS

Each specification included in the *XtActions* file modifies a key setting for the editor that *xedit* uses. When defining key specifications, you must use the following resource specification: text.EventBindings: .XtActions

Each key specification assigns an editor command to a named key and/or mouse combination and has the format:

key: *function*

key	Specifies the key or mouse button that is used to invoke the named function.
function	Specifies the function to be invoked when the named key is pressed.

For more information about specifications in the *XtActions* file, see *X(1)*.

FILES

~/.XtActions
/usr/lib/X11/.XtActions

SEE ALSO

X(1), xrdb(1)

RESTRICTIONS

Large numbers of certain edit functions (for example, Undo or More) tend to degrade performance over time. If there is a noticeable decrease in response time, save and reload the file.

BUGS

It is not clear how to select a line number for the *Jump* command.

The string searches don't work properly.

COPYRIGHT

Copyright 1988, Digital Equipment Corporation.

NAME

xev - print contents of X events

SYNOPSIS

xev [–display *displayname*] [–geometry *geom*]

DESCRIPTION

Xev creates a window and then asks the X server to send it notices called *events* whenever anything happens to the window (such as being moved, resized, typed in, clicked in, etc.). It is useful for seeing what causes events to occur and to display the information that they contain.

OPTIONS

—display *display*

This option specifies the X server to contact.

—geometry *geom*

This option specifies the size and/or location of the window.

SEE ALSO

X(1), xwininfo(1), xdpyinfo(1), Xlib Programmers Manual, X Protocol Specification

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xeyes – watch over your shoulder

SYNOPSIS

xeyes [-option ...]

DESCRIPTION

Xeyes watches what you do and reports to the Boss.

OPTIONS

–fg *foreground color*
: choose a different color for the pupil of the eyes.

–bg *background color*
: choose a different color for the background.

–outline *outline color*
: choose a different color for the outline of the eyes.

–center *center color*
: choose a different color for the center of the eyes.

–backing *{ WhenMapped Always NotUseful }*
: selects an appropriate level of backing store.

–geometry *geometry*
: define the initial window geometry; see *X(1)*.

–display *display*
: specify the display to use; see *X(1)*.

–bd *border color*
: choose a different color for the window border.

–bw *border width*
: choose a different width for the window border.

–shape uses the SHAPE extension to shape the window.

SEE ALSO

X(1), X Toolkit documentation

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Keith Packard, MIT X Consortium

NAME

xfd - font displayer for X

SYNOPSIS

xfd [-options ...] -fn *fontname*

OPTIONS

Xfd accepts all of the standard toolkit command line options along with the additional options listed below:

—fn *font* This option specifies the font to be displayed.

—box This option indicates that a box outlining the area that would be filled with background color by an ImageText request.

—center This option indicates that each glyph should be centered in its grid.

—start *number*
This option specifies the glyph index of the upper left hand corner of the grid. This is used to view characters at arbitrary locations in the font. The default is 0.

—bc *color*
This option specifies the color to be used if ImageText boxes are drawn.

DESCRIPTION

The *xfd* utility creates a window containing the name of the font being displayed, a row of command buttons, several lines of text for displaying character metrics, and a grid containing one glyph per cell. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left will be character number 0 unless the -start option has been supplied in which case the character with the number given in the -start option will be used.

The characters are displayed in a grid of boxes, each large enough to hold any single character in the font. Each character glyph is drawn using the PolyText16 request (used by the *Xlib* routine **XDrawString16**). If the *-box* option is given, a rectangle will be drawn around each character, showing where an ImageText16 request (used by the *Xlib* routine **XDrawImage-String16**) would cause background color to be displayed.

The origin of each glyph is normally set so that the character is drawn in the upper left hand corner of the grid cell. However, if a glyph has a negative left bearing or an unusually large ascent, descent, or right bearing (as is the case with *cursor* font), some character may not appear in their own grid cells. The *-center* option may be used to force all glyphs to be centered in their respective cells.

All the characters in the font may not fit in the window at once. To see the next page of glyphs, press the *Next* button at the top of the window. To see the previous page, press *Prev*. To exit *xfd*, press *Quit*.

Individual character metrics (index, width, bearings, ascent and descent) can be displayed at the top of the window by pressing on the desired character.

The font name displayed at the top of the window is the full name of the font, as determined by the server. See *xlsfonts* for ways to generate lists of fonts, as well as more detailed summaries of their metrics and properties.

X DEFAULTS

To be written.

SEE ALSO

X(1), xlsfonts(1), xrdb(1)

BUGS

The program should skip over pages full of non-existent characters.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium; previous program of the same name by Mark Lillibridge, MIT Project Athena.

NAME

xfig – Facility for Interactive Generation of figures under X11

SYNOPSIS

xfig [**-ri[ght]**] [**-le[ft]**] [**-L[andscape]**] [**-P[ortrait]**] [**-w[idth]** *inches*] [**-h[eight]** *inches*] [**-no[track]**] [**-tr[ack]**] [*file*]

DESCRIPTION

Xfig is a menu-driven tool that allows the user to draw and manipulate objects interactively in an X window. It runs under X version 11 release 2 and requires a three-button mouse. *File* specifies the name of a file to be edited. The description of objects in the file will be read at the start of *xfig*.

The output from *xfig* can be printed in several ways

troff - f2p (*xfig* to *pic*(1) translator, also known by its previous name *ftop*(1L)) is used to translate *xfig* files into *pic*(1) language. The resulting file may then be processed in the same maner as any other *pic* file.

postscript - f2ps (*xfig* to *postscript* translator) is used to produce a *postscript* file from an *xfig* file. The *postscript* file can be sent directly to a postscript printer.

LaTeX - fig2latex (*xfig* to *LaTeX* translator) produces a *LaTeX* file from an *xfig* file. This file contains *LaTeX* picture environment commands and can be processed along with other *LaTeX* commands.

PiCTeX - fig2tex (*xfig* to *PiCTeX* translator) produces a *PiCTeX* file from an *xfig* file. This file contains macros that can be used with the *PiCTeX* environment under *TeX* or *LaTeX*.

OPTIONS

—ri Change the position of the panel window to the right of the canvas window (default: left).

—le Change the position of the panel window to the left of the canvas window.

—L Make *xfig* come up in landscape mode (10" x 7.5").

—P Make *xfig* come up in portrait mode (7.5" x 10"). This is the default.

-w *inches*
Make *xfig* come up *inches* wide.

-h *inches*
Make *xfig* come up *inches* high.

-tr Turn on cursor (mouse) tracking arrows.

-no Turn off cursor (mouse) tracking arrows.

GRAPHICAL OBJECTS

The objects in *xfig* are divided into **primitive objects** and **compound object**. The primitive objects are: *ARC*, *CIRCLE*, *CLOSED SPLINE*, *ELLIPSE*, *POLYLINE*, *POLYGON*, *SPLINE*, and *TEXT*. A primitive can be moved, rotated, flipped, copied or erased. A compound object is composed of primitive objects. The primitive objects that constitute a compound can not be individually modified, but they can be manipulated as an entity; a compound can be moved, rotated, flipped, copied or erased. An extra function that can be applied to a compound object is **scaling**, which is not available for primitive objects.

DISPLAY WINDOWS

Five windows comprise the display area of *xfig*: the top ruler, the side ruler, the panel window, the message window, and the canvas window. The message window always appears below the others; it is the area in which messages are sent and received. The panel window can be placed to the left or right of the the canvas window (default: left).

POP-UP MENU

The pop-up menu appears when the right mouse button is pressed with the cursor positioned within the canvas window. Positioning the cursor over the desired menu entry and releasing the button selects a menu entry.

There are a number of file accessing functions in the pop-up menu. Most of the time when one of these functions is selected, the user will be asked for a file name. If the specified file can be located and the access permission are granted, *xfig* will carry out the function. However in case things go wrong, *xfig* will abort the function and printed the causes on the message window.

Undo Undo the last object creation or modification.

Redisplay
Redraw the canvas.

Remove all
Remove all objects on the canvas window (can be undone).

Edit file ...
The current contents of the canvas are cleared and objects are read from the specified file. The user will be asked for a file name. This file will become the current file.

Save Save the current contents of the canvas in the current file. If no file is being edited, the user will be asked for a file name as in the "Save in ..." function.

Read file ...
Read objects from the specified file and merge them with objects already shown on the canvas. (The user will be asked for a file name.)

Save as ...
Save objects on the screen into a file specified by the user. (The user will be asked for a file name.)

Status Show the name of the current file and directory.

Change Directory
Change the working directory. Any file name without a full path name will employ the current working directory.

Save & Exit
Save the objects in the current file and exit from *xfig*. If there is no current file, the user will be asked for a file name. No confirmation will be asked.

Quit Exit from *xfig*, discarding all objects. The user will be asked to confirm the action, by clicking the left button.

Save as BITMAP ...
Create a bitmap picture of the drawings for use with other tools (for example, for use as an icon). The smallest rectangular area of pixels that encompasses the figure is written to the named file (the user will be asked for a file name) from top row to bottom and left to right (in Sun raster format). Only *TEXT* objects that are parts of compound objects will be treated as parts of the picture; other texts are saved as objects in *xfig* format following the bitmap data. The coordinates of these text objects can be used to identify locations on the bitmap.

PANEL WINDOW MANIPULATION FUNCTIONS

Icons in the panel window represent object manipulation functions, modes and other drawing or modification aids. Manipulation functions are selected by positioning the cursor over it and clicking the left mouse button. The selected icon is highlighted, and a message describing its function appears in the message window.

The left and middle buttons are used to creat and modify objects in the canvas window. Most actions start with clicking of the left button and end with clicking of the right button. There is no need to hold down a button while positioning the cursor.

PANEL WINDOW COMMAND DESCRIPTIONS

Entries in the panel window can be classified into two categories: object creation/modification/removal commands (only one of which may be active at any one time), and drawing aids (which act as toggle switches). There are two ways for drawing circles, two for ellipses, two for splines and two for closed splines. There are two basic splines. One is the interpolated spline in which the spline pass thorough the entered points (knots). The other is the normal spline in which on control points are passed by the spline (except for the two end points in the open spline).

OBJECT CREATION/MODIFICATION/REMOVAL

Multiple commands are grouped thematically in the following descriptions (which is listed alphabetically).

ADD/DELETE ARROWS
: Add or delete arrow heads for *POLYLINE*, *POLYGON*, *SPLINE* or *CLOSED SPLINE* objects (points of a *BOX* can not be added or deleted).

ADD/DELETE POINTS
: Add or delete points for *POLYLINE*, *POLYGON*, *SPLINE* or *CLOSED SPLINE* objects (points of a *BOX* can not be added or deleted).

ARC Create an arc. Specify three points using the left button.

BOX Create rectangular boxes. Start with the left button and terminate with the right button.

BREAK COMPOUND
: Break a compound object to allow manipulation of its component parts. Click the left button on the bounding box of the compound object.

CIRCLE
: Create circles by specifying their radii or diameters. Click the left button on the canvas window, move the cursor until the desired radius or diameter is reached, then click the middle button to terminate. The circle will be drawn after the pressing of the middle button.

CLOSED INTERPOLATED SPLINE
: Create closed or periodic splines. The function is similar to *POLYGON* except that a closed interpolated spline is drawn. The spline will pass through the points (knots).

CLOSED SPLINE
Create closed or periodic spline objects. The function is similar to *POLYGON* except that a closed spline will be drawn instead of polygon. The entered points are just control points; i.e., the spline will not pass any of these points.

COPY Copy object. Click the left button over part of the object to be copied (for *CIRCLE* and *ELLIPSE* objects, position on their circumferences). Drag the object to the desired position and click the middle button. This function as well as the following three functions (*MOVE, MOVE POINT, REMOVE*) will cause point markers (manipulation aids) to be shown on the canvas window. There are no markers for *CIRCLE* or *ELLIPSE* objects.

ELLIPSE
Create ellipses using the same procedure as for the drawing of circles.

GLUE Glue the objects within a bounding box into a compound object (the bounding box itself is not part of the figure; it is a visual aid for manipulating the compound).

INTERPOLATED SPLINE
Create (cubic spline) spline objects. Enter control vectors in the same way as for creation of a *POLYLINE* object. At least three points (two control vectors) must be entered. The spline will pass through the entered points.

MOVE Move objects in the same way as in *COPY.*

MOVE POINT
Modify the position of points of *POLYLINE, BOX, POLYGON, ELLIPSE, ARC and SPLINE objects. Click the left button over the desired point, reposition the point, and click the middle button. Note that BOX and POLYGON objects are internally stored as POLYLINE objects, and therefore moving certain points may open these objects.*

POLYGON
Same as *POLYLINE* except that a line segment is drawn connecting the first and last points entered.

POLYLINE
Create polylines (line segments connecting a sequence of points). Enter points by clicking the left button at the desired positions on the canvas window. Click the middle button to terminate.

REMOVE
Remove (or delete) objects.

SCALE COMPOUND
Only compound objects can be scaled. Click the left button on a corner of the bounding box, stretch the bounding box to the desired size, and click the middle button. Or click the left button on a side of the bounding box, stretch that side to the desired size, and click the middle button.

SPLINE Create (quadratic spline) spline objects. Enter control vectors in the same way as for creation of a *POLYLINE* object. At least three points (two control vectors) must be entered. The spline will pass only the two end points.

TEXT Create text strings. Click the left button at the desired position on the canvas window, then enter text from the keyboard. A DEL or ^H (backspace) will delete a character, while a ^U will kill the entire line. Terminate by clicking the middle button or typing the return key. To edit text, click on an existing text string with the left button. Insertion of characters will take place at that point.

TURN Turn *POLYGON* into a *CLOSED INTERPOLATED SPLINE* object, or turn *POLYLINE* into a *INTERPOLATED SPLINE* object.

DRAWING AIDS
Drawing aids act as toggle switches. More than one can be selected at a time (except for *GRID* and the line drawing modes).

AUTO FORWARD/BACKWARD ARROW
Automatically add forward/backward arrow heads to *POLYLINE*, *SPLINE* or *ARC* objects.

FLIP Invert the object (middle button) or produce a mirror-image copy of an object (left button). Point to part of the object ("the handle"), click the appropriate button.

GRID Display either the quarter- or half-inch grids (left button).

MAGNET
Round points to the nearest 1/16 of an inch. This affects every function, and is provided as an alignment aid.

UNRESTRICTED
Allow lines to be drawn with any slope.

MANHATTAN
Enforce drawing of lines in the horizontal and vertical direction only. Both *MANHATTAN* and *MOUNTAIN* can be turned on simultaneously. The creations of *POLYGON*, *POLYLINE* and

SPLINE objects are affected by these two modes.

MOUNTAIN
Enforce drawing of only diagonal lines. Both *MANHATTAN* and *MOUNTAIN* can be turned on simultaneously. The creations of *POLYGON, POLYLINE* and *SPLINE* objects are affected by these two modes.

MANHATTAN MOUNTAIN
Allow lines to be drawn at any slope allowed when in MOUNTIAIN or MANHATTAN modes.

LATEX LINE
Allow lines to be drawn only at slopes which can be handled by LaTeX picture environment lines: slope = x/y, where x,y are integers in the range [-6,6].

LATEX VECTOR
Allow lines to be drawn only at slopes which can be handled by LaTeX picture environment vectors: slope = x/y, where x,y are integers in the range [-4,4].

ROTATE
Rotate the object (middle button) or copy (left button) +90 degrees.

SOLID/DASHED LINE STYLE
Toggle between solid and dashed line styles. The dash length is fixed at 0.05 inch.

X DEFAULTS

The overall widget name(Class) is xfig.fig(Fig.TopLevelShell). This set of resources correspond to the command line arguments:

trackCursor	(boolean:on) -track and -notrack arguments
justify	(boolean:false) -right and -left arguments
landscape	(boolean:false) -Landscape and -Portrait arguments
debug	(boolean:off) -debug arguments
width	(integer:7.5 or 10 inches) -width argument
height	(integer:10 or 7.5 inches) -height argument
reverseVideo	(boolean:off) -inverse argument

These arguments correspond to the widgets which make up *xfig*.

overall window	form(Form)
side panel	form.panel(Form.Box)
icons	form.panel.button(Form.Box.Command)
top ruler	form.truler(Form.Label)
side ruler	form.sruler(Form.Label)
canvas	form.canvas(Form.Label)
message window	form.message(Form.Command)
menu	form.popup_menu.menu(Form.OverrideShell.Box)
menu title	form.popup_menu.menu.title(Form.OverrideShell.Box.Label)
menu items	form.popup_menu.menu.pane(Form.OverrideShell.Box.Command)

For example, to set the background of the panel to blue the resource would be:
xfig*form.panel.background: blue

NOTE: The font used in the canvas cannot be changed at this time.

BUGS

Text strings will appear differently on hard copy, because the display fonts are fixed-width fonts while the fonts used by the typesetter systems are variable-width fonts.

A double quote in a text string should be preceded by a back slash if the it is to be printed through *pic*(1).

Objects that extend beyond the canvas window may cause image shrinkage in hard copy printed by *pic*(1), since it will try to fit every object onto a single 8.5" x 11" page.

Ellipses which are too narrow may cause *xfig* to loop forever.

Objects which are created while one of the *grids* is on may appear ragged. This can be corrected by selecting *Redisplay* from the pop-up menu.

The X11 cursors are not the original ones but chosen from X11's cursor font.

SEE ALSO

Brian W. Kernighan *PIC - A Graphics Language for Typesetting User Manual*
ditroff(1), f2p(1), f2ps(1), fig2latex(1), fig2tex(1), pic(1), troff(1), tex(1), latex(1)

ACKNOWLEDGEMENT

Many thanks goes to Professor Donald E. Fussell who inspired the creation of this tool.

AUTHORS

Original author:
Supoj Sutanthavibul
University of Texas at Austin
(supoj@sally.utexas.edu)

Manual page modified by:
R. P. C. Rodgers
UCSF School of Pharmacy
San Francisco, CA 94118

The LaTeX line drawing modes were contributed by:
Frank Schmuck
Cornell University

X11 port by:
Ken Yap
Rochester
(ken@cs.rochester.edu)

Variable window sizes, cleanup of X11 port, right hand side panel under X11, X11 manual page provided by:
Dana Chee
Bellcore
(dana@bellcore.com)

Cleanup of color port to X11 by:
John T. Kohl
MIT
(jtkohl@athena.mit.edu)

NAME

xfontsel - point & click interface for selecting X11 font names

SYNTAX

xfontsel [-*toolkitoption* ...] [**-pattern** *fontname*] [**-print**] [**-sample** *text*]

DESCRIPTION

The *xfontsel* application provides a simple way to display the fonts known to your X server, examine samples of each, and retrieve the X Logical Font Description ("XLFD") full name for a font.

If **-pattern** is not specified, all fonts with XLFD 14-part names will be selectable. To work with only a subset of the fonts, specify **-pattern** followed by a partially or fully qualified font name; e.g., ‘‘-pattern *medium*’’ will select that subset of fonts which contain the string ‘‘medium’’ somewhere in their font name. Be careful about escaping wildcard characters in your shell.

If **-print** is specified on the command line the selected font specifier will be written to standard output when the *quit* button is activated. Regardless of whether or not **-print** was specified, the font specifier may be made the PRIMARY (text) selection by activating the *select* button.

The **-sample** option specifies the sample text to be used to display the selected font, overriding the default.

INTERACTIONS

Clicking any pointer button in one of the XLFD field names will pop up a menu of the currently-known possibilities for that field. If previous choices of other fields were made, only values for fonts which matched the previously selected fields will be selectable; to make other values selectable, you must deselect some other field(s) by choosing the ‘‘*’’ entry in that field. Unselectable values may be omitted from the menu entirely as a configuration option; see the **ShowUnselectable** resource, below. Whenever any change is made to a field value, *xfontsel* will assert ownership of the PRIMARY_FONT selection. Other applications (see, e.g., *xterm*) may then retrieve the selected font specification.

Clicking the left pointer button in the *select* widget will cause the currently selected font name to become the PRIMARY text selection as well as the PRIMARY_FONT selection. This then allows you to paste the string into other applications. The **select** button remains highlighted to remind you of this fact, and de-highlights when some other application takes the PRIMARY selection away. The *select* widget is a toggle; pressing it when it is highlighted will cause *xfontsel* to release the selection ownership and de-highlight the widget. Activating the *select* widget twice is the only way to cause *xfontsel* to release the PRIMARY_FONT selection.

RESOURCES

The application class is **XFontSel**. Most of the user-interface is configured in the app-defaults file; if this file is missing a warning message will be printed to standard output and the resulting window will be nearly incomprehensible.

Most of the significant parts of the widget hierarchy are documented in the app-defaults file (normally /usr/lib/X11/app-defaults/XFontSel).

Application specific resources:

cursor (class **Cursor**)
Specifies the cursor for the application window.

pattern (class **Pattern**)
Specifies the font name pattern for selecting a subset of available fonts. Equivalent to the **-pattern** option. Most useful patterns will contain at least one field delimiter; e.g. "*-m-*" for monospaced fonts.

printOnQuit (class **PrintOnQuit**)
If *True* the currently selected font name is printed to standard output when the quit button is activated. Equivalent to the **-print** option.

Widget specific resources:

showUnselectable (class **ShowUnselectable**)
Specifies, for each field menu, whether or not to show values that are not currently selectable, based upon previous field selections. If shown, the unselectable values are clearly identified as such and do not highlight when the pointer is moved down the menu. The full name of this resource is **fieldN.menu.options.showUnselectable**, class **MenuButton.SimpleMenu.Options.ShowUnselectable**; where N is replaced with the field number (starting with the left-most field numbered 0). The default is True for all but field 11 (average width of characters in font) and False for field 11. If you never want to see unselectable entries, '*menu.options.showUnselectable:False' is a reasonable thing to specify in a resource file.

FILES

$XFILESEARCHPATH/XFontSel

SEE ALSO

xrdb(1)

BUGS

Sufficiently ambiguous patterns can be misinterpreted and lead to an initial selection string which may not correspond to what the user intended and which may cause the initial sample text output to fail to match the proffered string. Selecting any new field value will correct the sample output, though possibly resulting in no matching font.

Should be able to return a FONT for the PRIMARY selection, not just a STRING.

Any change in a field value will cause *xfontsel* to assert ownership of the PRIMARY_FONT selection. Perhaps this should be parameterized.

When running on a slow machine, it is possible for the user to request a field menu before the font names have been completely parsed. An error message indicating a missing menu is printed to stderr but otherwise nothing bad (or good) happens.

COPYRIGHT

AUTHOR

Ralph R. Swick, Digital Equipment Corporation/MIT Project Athena

NAME

xgc - X graphics demo

SYNOPSIS

xgc [*-toolkitoption* ...]

DESCRIPTION

The *xgc* program demonstrates various features of the X graphics primitives. Try the buttons, see what they do; we haven't the time to document them, perhaps you do?

OPTIONS

Xgc accepts all of the standard X Toolkit command line options.

X DEFAULTS

This program accepts the usual defaults for toolkit applications.

ENVIRONMENT

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1)

BUGS

This program isn't really finished yet.

COPYRIGHT

AUTHORS

Dan Schmidt, MIT

NAME

xhost - server access control program for X

SYNOPSIS

xhost [[+-]hostname ...]

DESCRIPTION

The *xhost* program is used to add and delete hosts to the list of machines that are allowed to make connections to the X server. This provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment, although it does limit the worst abuses. Environments which require more sophisticated measures should use the hooks in the protocol for passing authentication data to the server.

The server initially allows network connections only from programs running on the same machine or from machines listed in the file */etc/X*.hosts* (where * is the display number of the server). The *xhost* program is usually run either from a startup file or interactively to give access to other users.

Hostnames that are followed by two colons (::) are used in checking DECnet connections; all other hostnames are used for TCP/IP connections.

OPTIONS

Xhost accepts the following command line options described below. For security, the options that effect access control may only be run from the same machine as the server.

[+]*hostname*
: The given *hostname* (the plus sign is optional) is added to the list of machines that are allowed to connect to the X server.

–*hostname*
: The given *hostname* is removed from the list of machines that are allowed to connect to the server. Existing connections are not broken, but new connection attempts will be denied. Note that the current machine is allowed to be removed; however, further connections (including attempts to add it back) will not be permitted. Resetting the server (thereby breaking all connections) is the only way to allow local connections again.

+
: Access is granted to everyone, even if they aren't on the list of allowed hosts (i.e. access control is turned off).

–
: Access is restricted to only those machines on the list of allowed hosts (i.e. access control is turned on).

nothing
: If no command line arguments are given, the list of hosts that are allowed to connect is printed on the standard output along with a message indicating whether or not access control is currently enabled. This is the only option that may be used from machines

other than the one on which the server is running.

FILES

/etc/X*.hosts

SEE ALSO

X(1), Xserver(1)

ENVIRONMENT

DISPLAY

to get the default host and display to use.

BUGS

You can't specify a display on the command line because **–display** is a valid command line argument (indicating that you want to remove the machine named *''display''* from the access list).

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHORS

Bob Scheifler, MIT Laboratory for Computer Science,
Jim Gettys, MIT Project Athena (DEC).

NAME

xinit - X Window System initializer

SYNOPSIS

xinit [[client] options] [-- [server] [display] options]

DESCRIPTION

The *xinit* program is used to start the X Window System server and a first client program (usually a terminal emulator) on systems that cannot start X directly from */etc/init* or in environments that use multiple window systems. When this first client exits, *xinit* will kill the X server and then terminate.

If no specific client program is given on the command line, *xinit* will look for a file in the user's home directory called *.xinitrc* to run as a shell script to start up client programs. If no such file exists, *xinit* will use the following as a default:

```
xterm -geometry +1+1 -n login -display :0
```

If no specific server program is given on the command line, *xinit* will look for a file in the user's home directory called *.xserverrc* to run as a shell script to start up the server. If no such file exists, *xinit* will use the following as a default:

```
X :0
```

Note that this assumes that there is a program named *X* in the current search path. However, servers are usually named *Xdisplaytype* where *displaytype* is the type of graphics display which is driven by this server. The site administrator should, therefore, make a link to the appropriate type of server on the machine, or create a shell script that runs *xinit* with the appropriate server.

An important point is that programs which are run by *.xinitrc* and by *.xserverrc* should be run in the background if they do not exit right away, so that they don't prevent other programs from starting up. However, the last long-lived program started (usually a window manager or terminal emulator) should be left in the foreground so that the script won't exit (which indicates that the user is done and that *xinit* should exit).

An alternate client and/or server may be specified on the command line. The desired client program and its arguments should be given as the first command line arguments to *xinit*. To specify a particular server command line, append a double dash (--) to the *xinit* command line (after any client and arguments) followed by the desired server comand.

Both the client program name and the server program name must begin with a slash (/) or a period (.). Otherwise, they are treated as an arguments to be appended to their respective startup lines. This makes it possible to add arguments (for example, foreground and background colors) without having to retype the whole command line.

If an explicit server name is not given and the first argument following the double dash (--) is a colon followed by a digit, *xinit* will use that number as the display number instead of zero. All remaining arguments are appended to the server command line.

EXAMPLES

Below are several examples of how command line arguments in *xinit* are used.

xinit
This will start up a server named *X* and run the user's *.xinitrc*, if it exists, or else start an *xterm.*

xinit -- /usr/bin/X11/Xqdss :1
This is how one could start a specific type of server on an alternate display.

xinit -geometry =80x65+10+10 -fn 8x13 -j -fg white -bg navy
This will start up a server named *X*, and will append the given arguments to the default *xterm* command. It will ignore *.xinitrc*.

xinit -e widgets -- ./Xsun -l -c
This will use the command *./Xsun -l -c* to start the server and will append the arguments *-e widgets* to the default *xterm* command.

xinit /usr/ucb/rsh fasthost cpupig -display ws:1 -- :1 -a 2 -t 5
This will start a server named *X* on display 1 with the arguments *-a 2 -t 5*. It will then start a remote shell on the machine **fasthost** in which it will run the command *cpupig*, telling it to display back on the local workstation.

Below is a sample *.xinitrc* that starts a clock, several terminals, and leaves the window manager running as the "last" application. Assuming that the window manager has been configured properly, the user then chooses the "Exit" menu item to shut down X.

```
xrdb -load $HOME/.Xres
xsetroot -solid gray &
xclock -g 50x50-0+0 -bw 0 &
xload -g 50x50-50+0 -bw 0 &
xterm -g 80x24+0+0 &
xterm -g 80x24+0-0 &
uwm
```

Sites that want to create a common startup environment could simply create a default *.xinitrc* that references a site-wide startup file:

```
#!/bin/sh
. /usr/local/lib/site.xinitrc
```

Another approach is to write a script that starts *xinit* with a specific shell script. Such scripts are usually named *x11*, *xstart*, or *startx* and are a convenient way to provide a simple interface for novice users:

```
#!/bin/sh
xinit /usr/local/bin/startx -- /usr/bin/X11/Xhp :1
```

ENVIRONMENT VARIABLES

DISPLAY
This variable gets set to the name of the display to which clients should connect.

XINITRC
This variable specifies an init file containing shell commands to start up the initial windows. By default, *.xinitrc* in the home directory will be used.

SEE ALSO

X(1), Xserver(1), xterm(1), xrdb(1)

COPYRIGHT

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science

NAME

xkill - kill a client by its X resource

SYNOPSIS

xkill [–display *displayname*] [–id *resource*] [–button number] [–frame] [–all]

DESCRIPTION

Xkill is a utility for forcing the X server to close connections to clients. This program is very dangerous, but is useful for aborting programs that have displayed undesired windows on a user's screen. If no resource identifier is given with *-id*, *xkill* will display a special cursor as a prompt for the user to select a window to be killed. If a pointer button is pressed over a non-root window, the server will close its connection to the client that created the window.

OPTIONS

–display *displayname*
This option specifies the name of the X server to contact.

–id *resource*
This option specifies the X identifier for the resource whose creator is to be aborted. If no resource is specified, *xkill* will display a special cursor with which you should select a window to be kill.

–button *number*
This option specifies the number of pointer button that should be used in selecting a window to kill. If the word "any" is specified, any button on the pointer may be used. By default, the first button in the pointer map (which is usually the leftmost button) is used.

–all This option indicates that all clients with top-level windows on the screen should be killed. *Xkill* will ask you to select the root window with each of the currently defined buttons to give you several chances to abort. Use of this option is highly discouraged.

–frame This option indicates that xkill should ignore the standard conventions for finding top-level client windows (which are typically nested inside a window manager window), and simply believe that you want to kill direct children of the root.

XDEFAULTS

Button Specifies a specific pointer button number or the word "any" to use when selecting windows.

SEE ALSO

X(1), xwininfo(1), XKillClient and XGetPointerMapping in the Xlib Programmers Manual, KillClient in the X Protocol Specification

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium
Dana Chee, Bellcore

NAME

xload - load average display for X

SYNOPSIS

xload [-*toolkitoption* ...] [-scale *integer*] [-update *seconds*]

DESCRIPTION

The *xload* program displays a periodically updating histogram of the system load average. This program is nothing more than a wrapper around the Athena Load widget.

OPTIONS

Xload accepts all of the standard X Toolkit command line options along with the following additional options:

—scale *integer*

This option specifies the minimum number of tick marks in the histogram, where one division represents one load average point. If the load goes above this number, *xload* will create more divisions, but it will never use fewer than this number. The default is 1.

—update *seconds*

This option specifies the frequency in seconds at which *xload* updates its display. If the load average window is uncovered (by moving windows with a window manager or by the *xrefresh* program), the graph will be also be updated. The minimum amount of time allowed between updates is 5 seconds (which is also the default).

–hl *color*

This option specifies the color of the label and scale lines.

The following standard X Toolkit arguments are commonly used with *xload*:

–bd *color*

This option specifies the border color. The default is *black.*

–bg *color*

This option specifies the background color. The default is *white.*

–bw *pixels*

This option specifies the width in pixels of the border around the window. The default value is 2.

–fg *color*

This option specifies the graph color. The default color is *black.*

–fn *fontname*
This option specifies the font to be used in displaying the name of the host whose load is being monitored. The default is *6x10.*

–rv
This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry*
This option specifies the prefered size and postion of the window.

–display *display*
This option specifies the X server to contact.

–xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Load* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width)**
Specifies the width of the load average graph.

height (class **Height)**
Specifies the height of the load average graph.

update (class **Interval)**
Specifies the frequency in seconds at which the load should be redisplayed.

scale (class **Scale)**
Specifies the initial number of ticks on the graph. The default is 1.

minScale (class **Scale)**
Specifies the minimum number of ticks that will be displayed. The default is 1.

foreground (class **Foreground)**
Specifies the color for the graph. The default is *black* since the core default for background is *white.*

highlight (class **Foreground)**
Specifies the color for the text and scale lines. The default is the same as for the **foreground** resource.

label (class **Label**)
Specifies the label to use on the graph. The default is the hostname.

font (class **Font**)
Specifies the font to be used for the label. The default is *fixed*.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

SEE ALSO

X(1), xrdb(1), mem(4), Athena Load widget

DIAGNOSTICS

Unable to open display or create window. Unable to open /dev/kmem. Unable to query window for dimensions. Various X errors.

BUGS

This program requires the ability to open and read the special system file */dev/kmem*. Sites that do not allow general access to this file should make *xload* belong to the same group as */dev/kmem* and turn on the *set group id* permission flag.

Reading /dev/kmem is inherently non-portable. Therefore, the widget upon which this application is based must be ported to each new operating system.

Border color has to be explicitly specified when reverse video is used.

COPYRIGHT

AUTHORS

K. Shane Hartman (MIT-LCS) and Stuart A. Malone (MIT-LCS);
with features added by Jim Gettys (MIT-Athena), Bob Scheifler (MIT-LCS), and Tony Della Fera (MIT-Athena)

NAME

xlogo - X Window System logo

SYNOPSIS

xlogo [-*toolkitoption* ...]

DESCRIPTION

The *xlogo* program displays the X Window System logo. This program is nothing more than a wrapper around the Athena Logo widget.

OPTIONS

Xlogo accepts all of the standard X Toolkit command line options, of which the following are used most frequently:

–bg *color*
: This option specifies the color to use for the background of the window. The default is *white.* A correct color for the background is something like *maroon.*

–bd *color*
: This option specifies the color to use for the border of the window. The default is *black.*

–bw *number*
: This option specifies the width in pixels of the border surrounding the window.

–fg *color*
: This option specifies the color to use for displaying the logo. The default is *black.* A correct color for the background is something like *silver*, which you can approximate with a shade of gray, like #aa9.

–rv
: This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry*
: This option specifies the prefered size and position of the logo window.

–display *display*
: This option specifies the X server to contact.

–xrm *resourcestring*
: This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Logo* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
: Specifies the width of the logo. The default width is 100 pixels.

height (class **Height**)
: Specifies the height of the logo. The default height is 100 pixels.

foreground (class **Foreground**)
: Specifies the color for the logo. The default is *black* since the core default for background is *white.*

reverseVideo (class **ReverseVideo**)
: Specifies that the foreground and background should be reversed.

ENVIRONMENT

DISPLAY
: to get the default host and display number.

XENVIRONMENT
: to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1), xrdb(1), Athena Logo widget

COPYRIGHT

AUTHORS

Ollie Jones of Apollo Computer wrote the logo graphics routine, based on a graphic design by Danny Chong and Ross Chapman of Apollo Computer.

NAME

xlsatoms - list interned atoms defined on server

SYNOPSIS

xlsatoms [-options ...]

DESCRIPTION

Xlsatoms lists the interned atoms. By default, all atoms starting from 1 (the lowest atom value defined by the protocol) are listed until unknown atom is found. If an explicit range is given, *xlsatoms* will try all atoms in the range, regardless of whether or not any are undefined. used.

OPTIONS

–display *dpy*
: This option specifies the X server to which to connect.

–format *string*
: This option specifies a *printf*-style string used to list each atom <*value,name*> pair, printed in that order (*value* is an *unsigned long* and *name* is a *char* *). *Xlsatoms* will supply a newline at the end of each line. The default is *%ld\t%s*.

–range *[low]-[high]*
: This option specifies the range of atom values to check. If *low* is not given, a value of 1 assumed. If *high* is not given, *xlsatoms* will stop at the first undefined atom at or above *low*.

–name *string*
: This option specifies the name of an atom to list. If the atom does not exist, a message will be printed on the standard error.

SEE ALSO

X(1), Xserver(1), xprop(1)

ENVIRONMENT

DISPLAY
: to get the default host and display to use.

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xlsclients - list client applications running on a display

SYNOPSIS

xlsclients [-display *displayname*] [-a] [-l] [-m maxcmdlen]

DESCRIPTION

Xlsclients is a utility for listing information about the client applications running on a display. It may be used to generate scripts representing a snapshot of the the user's current session.

OPTIONS

—display *displayname*
: This option specifies the X server to contact.

—a
: This option indicates that clients on all screens should be listed. By default, only those clients on the default screen are listed.

—l
: This option indicates that a long listing showing the window name, icon name, and class hints in addition to the machine name and command string shown in the default listing.

—m *maxcmdlen*
: This option specifies the maximum number of characters in a command to print out. The default is 1000.

ENVIRONMENT

DISPLAY
: To get the default host, display number, and screen.

SEE ALSO

X(1), xwininfo(1), xprop(1)

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xlsfonts - server font list displayer for X

SYNOPSIS

xlsfonts [-options ...] [-fn pattern]

DESCRIPTION

Xlsfonts lists the fonts that match the given *pattern*. The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

−display *host:dpy*
: This option specifies the X server to contact.

−l[l[l]] This option indicates that medium, long, and very long listings, respectively, should be generated for each font.

−m This option indicates that long listings should also print the minimum and maximum bounds of each font.

−C This option indicates that listings should use multiple columns. This is the same as **-n 0**.

−1 This option indicates that listings should use a single column. This is the same as **-n 1**.

−w *width*
: This option specifies the width in characters that should be used in figuring out how many columns to print. The default is 79.

−n *columns*
: This option specifies the number of columns to use in displaying the output. By default, it will attempt to fit as many columns of font names into the number of character specified by **-w** *width*.

SEE ALSO

X(1), Xserver(1), xset(1), xfd(1)

ENVIRONMENT

DISPLAY
: to get the default host and display to use.

BUGS

Doing ``xlsfonts -l'' can tie up your server for a very long time. This is really a bug with single-threaded non-preemptible servers, not with this program.

COPYRIGHT

AUTHOR

Mark Lillibridge, MIT Project Athena; Jim Fulton, MIT X Consortium; Phil Karlton, SGI

NAME

xlswins - server window list displayer for X

SYNOPSIS

xlswins [-options ...] [windowid ...]

DESCRIPTION

Xlswins lists the window tree. By default, the root window is used as the starting point, although a specific window may be specified using the *-id* option. If no specific windows are given on the command line, the root window will be used.

OPTIONS

–display *displayname*
: This option specifies the X server to contact.

–l
: This option indicates that a long listing should be generated for each window. This includes a number indicating the depth, the geometry relative to the parent as well as the location relative to the root window.

–format *radix*
: This option specifies the radix to use when printing out window ids. Allowable values are: *hex*, *octal*, and *decimal*. The default is hex.

–indent *number*
: This option specifies the number of spaces that should be indented for each level in the window tree. The default is 2.

SEE ALSO

X(1), Xserver(1), xwininfo(1), xprop(1)

ENVIRONMENT

DISPLAY
: to get the default host and display to use.

BUGS

This should be integrated with xwininfo somehow.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xmag - magnify parts of the screen

SYNOPSIS

xmag [-option ...]

DESCRIPTION

The *xmag* program allows you to magnify portions of the screen. If no explicit region is specified, a square centered around the pointer is displayed indicating the area to be enlarged. Once a region has been selected, a window is popped up showing a blown up version of the region in which each pixel in the source image is represented by a small square of the same color. Pressing Button1 on the pointer in the enlargement window pops up a small window displaying the position, number, and RGB value of the pixel under the pointer until the button is released. Pressing the space bar or any other pointer button removes the enlarged image so that another region may be selected. Pressing ‘‘q’’, ‘‘Q’’, or ‘‘^C’’ in the enlargement window exits the program.

OPTIONS

—display *display*
: This option specifies the X server to use for both reading the screen and displaying the enlarged version of the image.

—geometry *geom*
: This option specifies the size and/or location of the enlargement window. By default, the size is computed from the size of the source region and the desired magnification. Therefore, only one of **–source** *size* and **–mag** *magfactor* options may be specified if a window size is given with this option.

—source *geom*
: This option specifies the size and/or location of the source region on the screen. By default, a 64x64 square centered about the pointer is provided for the user to select an area of the screen. The size of the source is used with the desired magnification to compute the default enlargement window size. Therefore, only one of **–geometry** *size* and **–mag** *magfactor* options may be specified if a source size is given with this option.

–mag *magfactor*
: This option specifies an integral factor by which the source region should be enlarged. The default magnification is 5. This is used with the size of the source to compute the default enlargement window size. Therefore, only one of **-geometry** *size* and **–source** *geom* options may be specified if a magnification factor is given with this option.

−bw *pixels*
: This option specifies the width in pixels of the border surrounding the enlargement window.

−bd *color*
: This option specifies the color to use for the border surrounding the enlargement window.

−bg *colororpixelvalue*
: This option specifies the name of the color to be used as the background of the enlargement window. If the name begins with a percent size (%), it is interpretted to be an absolute pixel value. This is useful when displaying large areas since pixels that are the same color as the background do not need to be painted in the enlargement. The default is to use the BlackPixel of the screen.

−fn *fontname*
: This option specifies the name of a font to use when displaying pixel values (used when Button1 is pressed in the enlargement window).

−z This option indicates that the server should be grabbed during the dynamics and the call to XGetImage. This is useful for ensuring that clients don't change their state as a result of entering or leaving them with the pointer.

X DEFAULTS

The *xmag* program uses the following X resources:

geometry (class **Geometry**)
: Specifies the size and/or location of the enlargement window.

source (class **Source**)
: Specifies the size and/or location of the source region on the screen.

magnification (class **Magnification**)
: Specifies the enlargement factor.

borderWidth (class **BorderWidth**)
: Specifies the border width in pixels.

borderColor (class **BorderColor**)
: Specifies the color of the border.

background (class **Background**)
: Specifies the color or pixel value to be used for the background of the enlargement window.

font (class **Font**)
Specifies the name of the font to use when displaying pixel values when the user presses Button1 in the enlargement window.

SEE ALSO

X(1), xwd(1)

BUGS

This program will behave strangely on displays that support windows of different depths.

Because the window size equals the source size times the magnification, you only need to specify two of the three parameters. This can be confusing.

Being able to drag the pointer around and see a dynamic display would be very nice.

Another possible interface would be for the user to drag out the desired area to be enlarged.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xman - Manual page display program for the X Window System.

SYNOPSIS

xman [-options ...]

DESCRIPTION

Xman is a manual page browser. The default size of the initial *xman* window is small so that you can leave it running throughout your entire login session. In the initial window there are three options: *Help* will pop up a window with on-line help, *Quit* will exit, and *Manual Page* will pop up a window with a manual page browser in it. You may pop up more than one manual page browser window from a single execution of *xman*.

For further information on using *xman* please read the on-line help information. The rest of this manual page will discuss customization of *xman*.

CUSTOMIZING XMAN

Xman allows customization of both the directories to be searched for manual pages, and the name that each directory will map to in the *Sections* menu. Xman determines which directories it will search by reading the *MANPATH* environment variable. If no *MANPATH* is found then the directory is /usr/man is searched on POSIX systems. This environment is expected to be a colon-separated list of directories for xman to search.

```
setenv MANPATH /mit/kit/man:/usr/man
```

By default, *xman* will search each of the following directories (in each of the directories specified in the users MANPATH) for manual pages. If manual pages exist in that directory then they are added to list of manual pages for the corresponding menu item. A menu item is only displayed for those sections that actually contain manual pages.

Directory	Section Name
---------	------------
man1	(1) User Commands
man2	(2) System Calls
man3	(3) Subroutines
man4	(4) Devices
man5	(5) File Formats
man6	(6) Games
man7	(7) Miscellaneous
man8	(8) Sys. Administration
manl	(l) Local
mann	(n) New
mano	(o) Old

For instance, a user has three directories in her manual path and each contain a directory called *man3*. All these manual pages will appear alphabetically sorted when the user selects the menu item called *(3) Subroutines*. If there is no directory called *mano* in any of the directories in her MANPATH, or there are no manual pages in any of the directories called *mano* then no menu item will be displayed for the section called *(o) Old*.

By using the *mandesc* file a user or system manager is able to more closely control which manual pages will appear in each of the sections represented by menu items in the *Sections* menu. This functionality is only available on a section by section basis, and individual manual pages may not be handled in this manner (Although generous use of symbolic links - ln(1) - will allow almost any configuration you can imagine).

The format of the mandesc file is a character followed by a label. The character determines which of the sections will be added under this label. For instance suppose that you would like to create an extra menu item that contains all programmer subroutines. This label should contain all manual pages in both sections two and three. The *mandesc* file would look like this:

```
2Programmer Subroutines
3Programmer Subroutines
```

This will add a menu item to the *Sections* menu that would bring up a listing of all manual pages in sections two and three of the Programmers Manual. Since the label names are *exactly* the same they will be added to the same section. Note, however, that the original sections still exist.

If you want to completely ignore the default sections in a manual directory then add the line:

```
no default sections
```

anywhere in your mandesc file. This keeps xman from searching the default manual sections *In that directory only*. As an example, suppose you want to do the same thing as above, but you don't think that it is useful to have the *System Calls* or *Subroutines* sections any longer. You would need to duplicate the default entries, as well as adding your new one.

```
no default sections
1(1) User Commands
2Programmer Subroutines
3Programmer Subroutines
4(4) Devices
```

5(5) File Formats
6(6) Games
7(7) Miscellaneous
8(8) Sys. Administration
l(l) Local
n(n) New
o(o) Old

Xman will read any section that is of the from *man<character>*, where <character> is an upper or lower case letter (they are treated distinctly) or a numeral (0-9). Be warned, however, that man(1) and catman(8) will not search directories that are non-standard.

COMMAND LINE OPTIONS

Xman supports all standard Toolkit command line arguments (see *X(1)*). The following additional arguments are supported.

-helpfile *filename*
: Specifies a helpfile to use other than the default.

-bothshown
: Allows both the manual page and manual directory to be on the screen at the same time.

-notopbox
: Starts without the Top Menu with the three buttons in it.

-geometry *WxH+X+Y*
: Sets the size and location of the Top Menu with the three buttons in it.

-pagesize *WxH+X+Y*
: Sets the size and location of all the Manual Pages.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xman*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Xman xman       (This widget is never used)
        TopLevelShell  topbox
                Form  form
                        Label  topLabel
                        Command  helpButton
                        Command  quitButton
                        Command  manpageButton
                TransientShell search
                        DialogWidgetClass dialog
                                Label label
                                Text value
                                Command manualPage
                                Command apropos
                                Command cancel
                TransientShell pleaseStandBy
                        Label label
        TopLevelShell manualBrowser
                Paned Manpage_Vpane
                        Paned horizPane
                                MenuButton options
                                MenuButton sections
                                Label manualBrowser
                        Viewport directory
                                List directory
                                List directory
                                .
                                . (one for each section,
                                . created "on the fly")
                                .
                        ScrollByLine manualPage
                SimpleMenu optionMenu
                        SmeBSB displayDirectory
                        SmeBSB displayManualPage
                        SmeBSB help
                        SmeBSB search
                        SmeBSB showBothScreens
                        SmeBSB removeThisManpage
                        SmeBSB openNewManpage
                        SmeBSB showVersion
                        SmeBSB quit
                SimpleMenu sectionMenu
                        SmeBSB <name of section>
                                .
                                . (one for each section)
                                .
                TransientShell search
                        DialogWidgetClass dialog
                                Label label
                                Text value
                                Command manualPage
```

```
                                        Command apropos
                                        Command cancel
                        TransientShell pleaseStandBy
                                Label label
                        TransientShell likeToSave
                                Dialog dialog
                                        Label label
                                        Text value
                                        Command yes
                                        Command no
                TopLevelShell help
                        Paned Manpage_Vpane
                                Paned horizPane
                                        MenuButton options
                                        MenuButton sections
                                        Label manualBrowser
                                ScrollByLine manualPage
                        SimpleMenu optionMenu
                                SmeBSB displayDirectory
                                SmeBSB displayManualPage
                                SmeBSB help
                                SmeBSB search
                                SmeBSB showBothScreens
                                SmeBSB removeThisManpage
                                SmeBSB openNewManpage
                                SmeBSB showVersion
                                SmeBSB quit
```

APPLICATION RESOURCES

xman has the following application-specific resources which allow customizations unique to *xman*.

manualFontNormal (Class **Font**)
The font to use for normal text in the manual pages.

manualFontBold (Class **Font**)
The font to use for bold text in the manual pages.

manualFontItalic (Class **Font**)
The font to use for italic text in the manual pages.

directoryFontNormal (Class **Font**)
The font to use for the directory text.

bothShown (Class **Boolean**)
Either 'true' or 'false', specifies whether or not you want both the directory and the manual page shown at start up.

directoryHeight (Class **DirectoryHeight**)
The height in pixels of the directory, when the directory and the manual page are shown simultaneously.

topCursor (Class **Cursor**)
The cursor to use in the top box.

helpCursor (Class **Cursor**)
The cursor to use in the help window.

manpageCursor (Class **Cursor**)
The cursor to use in the manual page window.

searchEntryCursor (Class **Cursor**)
The cursor to use in the search entry text widget.

pointerColor (Class **Foreground**)
This is the color of all the cursors (pointers) specified above. The name was chosen to be compatible with xterm.

helpFile (Class **File**) Use this rather than the system default helpfile.

topBox (Class **Boolean**)
Either 'true' or 'false', determines whether the top box (containing the help, quit and manual page buttons) or a manual page is put on the screen at start-up. The default is true.

verticalList (Class **Boolean**)
Either 'true' or 'false', determines whether the directory listing is vertically or horizontally organized. The default is horizontal (false).

GLOBAL ACTIONS

Xman defines all user interaction through global actions. This allows the user to modify the translation table of any widget, and bind any event to the new user action. The list of actions supported by *xman* are:

GotoPage(*page*) When used in a manual page display window this will allow the user to move between a directory and manual page display. The *page* argument can be either **Directory** or **ManualPage**.

Quit() This action may be used anywhere, and will exit xman.

Search(*type*, *action*) Only useful when used in a search popup, this action will cause the search widget to perform the named search type on the string in the search popup's value widget. This action will also pop down the search widget. The *type* argument can be either **Apropos**, **Manpage** or **Cancel**. If an *action* of **Open** is specified then xman will open a new manual page to display the results of the search, otherwise xman will attempt to display the results in the parent of the search popup.

PopupHelp() This action may be used anywhere, and will popup the help widget.

PopupSearch() This action may be used anywhere except in a help window. It will cause the search popup to become active and visible on the screen, allowing the user search for a manual page.

CreateNewManpage() This action may be used anywhere, and will create a new manual page display window.

RemoveThisManpage() This action may be used in any manual page or help display window. When called it will remove the window, and clean up all resources associated with it.

SaveFormattedPage(*action*)
This action can only be used in the "likeToSave" popup widget, and tells xman whether to **Save** or **Cancel** a save of the manual page that has just been formatted.

ShowVersion() This action may be called from any manual page or help display window, and will cause the informational display line to show the current version of xman.

FILES

<*manpath directory*>/man<*character*>

<*manpath directory*>/cat<*character*>

<*manpath directory*>/mandesc

/usr/lib/X11/app-defaults/Xman	specifies required resources
/tmp	*Xman* creates temporary files in /tmp for all unformatted man pages and all apropos searches.

SEE ALSO

X(1), X(8C), man(1), apropos(1), catman(8), Athena Widget Set

ENVIRONMENT

DISPLAY	the default host and display to use.
MANPATH	the search path for manual pages. Directories are separated by colons (e.g. /usr/man:/mit/kit/man:/foo/bar/man).
XENVIRONMENT	to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.
XAPPLRESDIR	A string that will have "Xman" appended to it. This string will be the full path name of a user app-defaults file to be merged into the resource database after the system app-defaults file, and before the resources that are attached to the display.

BUGS

There probably are some.

COPYRIGHT

Copyright 1988 by Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHORS

Chris Peterson, MIT X Consortium from the V10 version written by Barry Shein formerly of Boston University.

NAME

xmessage - X window system message display program.

SYNOPSIS

xmessage [-options ...] -m <message>

DESCRIPTION

Xmessage opens a window on the screen that will contain the text of a message from either the command line or stdin. This text may have a scroll bar along the left side to allow the user to browse through relatively long messages. Along the lower edge of the message is list of words with boxes around them, clicking the left mouse button on any of these "buttons" (words with boxes around them) will cause the message to go away. If there is more than one "button" then some state will be returned to the invoker of the xmessage process via a change of the exit status of the program.

This program serves two functions, firstly it can be a method for shell scripts to present the user with information much as 'echo' allows in a tty environment, as well as allowing the user you answer simple questinos. Secondly it allows much of the functionality of 'cat' again in a windowing verses tty environment.

It should be noted that this program is intended for short messages, and will be quite slow when asked to handle long files from stdin. Although xmessage can accept input from stdin, this input is not allowed to come from a tty, if this is attempted, and error message will be printed. If xmessage is executed with an incorrect argument then it will print a usage message to standard out, as well as to an xmessage window.

COMMAND LINE OPTIONS

These are the command line options that xmessage understands. Please note that some of these are inherited from the XToolkit and as the list of default toolkit options changes xman will follow.

-printlabel

This will case the program to print the label of the button pressed to standard output (stdout), I envision this to be useful when popping up a message to a friend, as in: "ready to go to lunch". This allows you to know if he clicked the yes or the no button.

-noscroll (-nsb)

The scroll bar is active on the text window by default, this causes it to be removed.

-buttons <button> <button> ...
This option will cause xmessage to create one button for each arguement the follows it until something starts with a '-'. The string passed to the button is the name of the Command button widget created and will be the default text displayed to the user. Since this is the name of the widget it may be used to change any of the Xresources associated with that button.

-message <word> <word> ...
This must be the last argument in the command list, as every argument after this one is assumed to be part of the message. There is no limit to the length of this message.

-geometery (height)x(width)+(x_offset)+(y_offset)
Sets the size and location of the window created by xmessage.

-bw <pixels>

-borderwidth <pixels>
Specifies the width of the border for all windows in xmessage.

-bd <color>

-bordercolor <color>
Specifies the color of the borders of all windows in xmessage.

-fg <color>

-foreground <color>
Specifies the foreground color to be used.

-bg <color>

-background <color>
Specifies the background color to be used.

-fn <font>

-font <font>
Specifies the font to use for all buttons and text.

-display <host:display[.screen]>
Specifies a display other than the default specified by the DISPLAY environment variable to use to use.

-name <name>
Specifies the name to use when retrieving resources.

-title <title>
Specifies the title of this application.

-xrm <resource>
Allows a resource to be specified on the command line.

WIDGET AND RESOURCE NAMES

Resource management is an important part of X Toolkit applications, and xmessage is not exception, all objects in xmessage can have many of their distinguishing characteristics changed by changing the resources associated with them, below is a brief list of the resources and what they modify.

foreground	- foreground color
background	- background color
width & height	- size
borderWidth	- border width
borderColor	- border color

In order to change the default values for the widget resources you need to have the names, thus, below I have specified the names of some of the most common widgets.

xmessage - (argv[0])	- shell widget that contains everything displayed.
text	- the text window.
<button name>	- each of the buttons. "okay" is default.

You can also reference Widgets by class, The important classes for this application are: Command and Text.

Here are a few example of how to string all this information together into a resource specification, that can be used on the command line with the -xrm flag, or added to your .Xresource file.

xmessage*Command.foreground: Blue	All command buttons will be blue.
xmessage*foreground: Blue	Everything in the xmessage window has a blue foreground.
xmessage*Text.border: Red	The text widget has a red border.

In addition Xmessage has a few specific application resources, that allow customizations that are specific to xmessage.

ScrollText
A Boolean reasource that determines whether you are allowed to scroll the text widget the default value is TRUE.

printLabel
A Boolean resource that determines whether or not the label of the buton pressed to exit the program is printed, default value is FALSE.

ERROR MESSAGES

Xmessage errors may be printed into their owm xmessage window, this invocation of xmessage has a different name. This allows its resources to be specified seperatly, the name of xmessage error program is xmessage_error.

EXIT STATUS

Xmessage will exit with status zero (0) when there is only one button in the list, and it is clicked to exit. If there is more than one button in the list then the exit status will corrospond the number of the button pressed, starting at one hundred and one (101) for the first button, and counting up. Zero (0) is not used because no button should have a prefered place over the others.

SEE ALSO

X(1), X(8C), echo(1), cat(1)

BUGS

There must be some, somewhere.

AUTHORS

Copyright 1988 by Massachusetts Institute of Technology.
Chris Peterson, MIT Project Athena

NAME

xmh — X interface to the MH message handling system

SYNOPSIS

xmh [-path *mailpath*] [-initial *foldername*] [-flag] [-*toolkitoption* ...]

DESCRIPTION

The *xmh* program provides a window-oriented user interface to the Rand *MH* Message Handling System. To actually do things with your mail, it makes calls to the *MH* package. Electronic mail messages may be composed, sent, received, replied to, forwarded, sorted, and stored in folders.

To specify an alternate collection of mail folders in which to process mail, use **-path** followed by the pathname of the alternate mail directory. The default mail path is the value of the Path component in $HOME/.mh_profile, or $HOME/Mail if the *MH* Path is not given. To specify an alternate folder which may receive new mail and is initially opened by *xmh*, use the **-initial** flag. The default initial folder is 'inbox'. The option **-flag** will cause *xmh* to attempt to change the appearance of its icon when new mail has arrived. These three options have corresponding application-specific resources, named **MailPath**, **InitialFolder**, and **MailWaitingFlag**, which can be used in a resource file. The standard toolkit command line options are given in *X(1)*.

Please don't be misled by the size of this document. It introduces many aspects of the Athena Widget Set, and provides extensive mechanism for customization of the user interface. *xmh* really is easy to use.

INSTALLATION

The current version of *xmh* requires that the user is already set up to use *MH*, version 6. To do so, see if there is a file called .mh_profile in your home directory. If it exists, check to see if it contains a line that starts with "Current-Folder". If it does, you've been using version 4 or earlier of *MH*; to convert to version 6, you must remove that line. (Failure to do so causes spurious output to stderr, which can hang *xmh* depending on your setup.)

If you do not already have a .mh_profile, you can create one (and everything else you need) by typing "inc" to the shell. You should do this before using *xmh* to incorporate new mail.

For more information, refer to the *mh(1)* documentation.

BASIC SCREEN LAYOUT

xmh starts out with a single window, divided into four main areas:

— Six buttons with pull-down command menus.

— A collection of buttons, one for each top level folder. New users of mh will have two folders, ''drafts'' and ''inbox''.

— A listing, or Table of Contents, of the messages in the open folder. Initially, this will show the messages in ''inbox''.

— A view of one of your messages. Initially this is blank.

XMH AND THE ATHENA WIDGET SET

xmh uses the X Toolkit Intrinsics and the Athena Widget Set. Many of the features described below (scrollbars, buttonboxes, etc.) are actually part of the Athena Widget Set, and are described here only for completeness. For more information, see the Athena Widget Set documentation.

SCROLLBARS

Some parts of the main window will have a vertical area on the left containing a grey bar. This area is a *scrollbar*. They are used whenever the data in a window takes up more space than can be displayed. The grey bar indicates what portion of your data is visible. Thus, if the entire length of the area is grey, then you are looking at all your data. If only the first half is grey, then you are looking at the top half of your data. The message viewing area will have a horizontal scrollbar if the text of the message is wider than the viewing area.

You can use the pointer in the scrollbar to change what part of the data is visible. If you click with the middle button, then the top of the grey area will move to where the pointer is, and the corresponding portion of data will be displayed. If you hold down the middle button, you can drag around the grey area. This makes it easy to get to the top of the data: just press with the middle, drag off the top of the scrollbar, and release.

If you click with button 1, then the data to the right of the pointer will scroll to the top of the window. If you click with pointer button 3, then the data at the top of the window will scroll down to where the pointer is.

BUTTONBOXES, BUTTONS, AND MENUS

Any area containing many words or short phrases, each enclosed in a rectangle or rounded boundary, is called a *buttonbox*. Each rectangle or rounded area is actually a button that you can press by moving the pointer onto it and pressing pointer button 1. If a given buttonbox has more buttons in it than can fit, it will be displayed with a scrollbar, so you can always scroll to the button you want.

Some buttons have pull-down menus. Pressing the pointer button while the pointer is over one of these buttons will pull down a menu. Holding the button down while moving the pointer over the menu, called dragging the pointer, will highlight each selectable item on the menu as the pointer passes over it. To select an item in the menu, release the pointer button while the item is highlighted.

ADJUSTING THE RELATIVE SIZES OF AREAS

If you're not satisfied with the sizes of the various areas of the main window, they can easily be changed. Near the right edge of the border between each region is a black box, called a *grip*. Simply point to that grip with the pointer, press a pointer button, drag up or down, and release. Exactly what happens depends on which pointer button you press.

If you drag with the middle button, then only that border will move. This mode is simplest to understand, but is the least useful.

If you drag with pointer button 1, then you are adjusting the size of the window above. *xmh* will attempt to compensate by adjusting some window below it.

If you drag with pointer button 3, then you are adjusting the size of the window below. *xmh* will attempt to compensate by adjusting some window above it.

All windows have a minimum and maximum size; you will never be allowed to move a border past the point where it would make a window have an invalid size.

PROCESSING YOUR MAIL

This section will define the concepts of the selected folder, current folder, selected message(s), current message, selected sequence, and current sequence. Each *xmh* command is introduced.

For use in customization, action procedures corresponding to each command are given; these action procedures can be used to customize the user interface, particularly the keyboard accelerators and the functionality of the buttons in the optional button box created by the application resource **CommandButtonCount**.

SELECTED FOLDER

A folder contains a collection of mail messages, or is empty.

The selected folder is whichever foldername appears in the bar above the folder buttons. Note that this is not necessarily the same folder that is being viewed. To change the selected folder, just press on the desired folder button; if that folder has subfolders, select a folder from the pull down menu.

The Table of Contents, or toc, lists the messages in the viewed folder. The title bar above the Table of Contents displays the name of the viewed folder.

The toc title bar also displays the name of the viewed sequence of messages within the viewed folder. Every folder has an "all" sequence, which contains all the messages in the folder, and initially the toc title bar will show "inbox:all".

FOLDER COMMANDS

The *folder* command menu contains commands of a global nature:

Open Folder
Display the data in the selected folder. Thus, the selected folder also becomes the viewed folder. The action procedure corresponding to this command is **XmhOpenFolder**([*foldername*]). It takes an optional argument as the name of a folder to select and open; if no folder is specified, the selected folder is opened. It may be specified as part of an event translation from a folder menu button or from a folder menu, or as a binding of a keyboard accelerator to any widget other than the folder menu buttons or the folder menus.

Open Folder in New Window
Displays the selected folder in an additional main window. Note, however, that you may not reliably display the same folder in more than one window at a time, although *xmh* will not prevent you from trying. The corresponding action is **XmhOpenFolder-InNewWindow()**.

Create Folder
Create a new folder. You will be prompted for a name for the new folder; to enter the name, move the pointer to the blank box provided and type. Subfolders are created by specifying the parent folder, a slash, and the subfolder name. For example, to create a folder named "xmh" which is a subfolder of an existing

folder named ``clients'', type ``clients/xmh''. Click on the Okay button when finished, or just type Return; click on Cancel to cancel this operation. The action corresponding to Create Folder is **XmhCreateFolder()**.

Delete Folder

Destroy the selected folder. You will be asked to confirm this action (see CONFIRMATION WINDOWS). Destroying a folder will also destroy any subfolders of that folder. The corresponding action is **XmhDeleteFolder()**.

Close Window

Exits *xmh*, after first confirming that you won't lose any changes; or, if selected from any additional *xmh* window, simply closes that window. The corresponding action is **XmhClose()**.

HIGHLIGHTED MESSAGES, SELECTED MESSAGES AND THE CURRENT MESSAGE

It is possible to highlight a set of adjacent messages in the area of the Table of Contents. To highlight a message, click on it with pointer button 1. To highlight a range of messages, click on the first one with pointer button 1 and on the last one with pointer button 3; or press pointer button 1, drag, and release. To extend a range of selected messages, use pointer button 3. To highlight all messages in the table of contents, click rapidly three times with pointer button 1. To cancel any selection in the table of contents, click rapidly twice.

The selected messages are the same as the highlighted messages, if any. If no messages are highlighted, then the selected messages are considered the same as the current message.

The current message is indicated by a '+' next to the message number. It usually corresponds to the message currently being viewed. When a message is viewed, the title bar above the view will identify the message.

TABLE OF CONTENTS COMMANDS

The *Table of Contents* command menu contains commands which operate on the open, or viewed folder.

Incorporate New Mail

Add any new mail received to your inbox folder, and set the current message to be the first new message. (This command is selectable only if ``inbox'' is the folder being viewed.) The corresponding action is

XmhIncorporateNewMail().

Commit Changes Execute all deletions, moves, and copies that have been marked in this folder. The corresponding action is **XmhCommitChanges()**.

Pack Folder Renumber the messages in this folder so they start with 1 and increment by 1. The corresponding action is **XmhPackFolder()**.

Sort Folder Sort the messages in this folder in chronological order. As a side effect, this also packs the folder. The corresponding action is **XmhSortFolder()**.

Rescan Folder Rebuild the list of messages. This can be used whenever you suspect that *xmh*'s idea of what messages you have is wrong. (In particular, this is necessary if you change things using straight *MH* commands without using *xmh*.) The corresponding action is **XmhForceRescan()**.

MESSAGE COMMANDS

The *Message* command menu contains commands which operate on the selected message(s), or if there are no selected messages, the current message.

Compose Message Composes a new message. A new window will be brought up for composition; a description of it is given in the COMPOSITION WINDOWS section below. This command does not affect the current message. The corresponding action is **XmhComposeMessage()**.

View Next Message View the first selected message. If no messages are highlighted, view the current message. If current message is already being viewed, view the first unmarked message after the current message. The corresponding action is **XmhViewNextMessage()**.

View Previous View the last selected message. If no messages are highlighted, view the current message. If current message is already being viewed, view the first unmarked message before the current message. The corresponding action is **XmhViewPrevious()**.

Mark Deleted Mark the selected messages for deletion. If no messages are highlighted, then this mark the current message for deletion and automatically display the next unmarked message. The corresponding action is **XmhMarkDeleted()**.

Mark Move Mark the selected messages to be moved into the current (selected) folder. (If the current folder is the same as the viewed folder, this command will just beep.) If no messages are highlighted, this will mark the current message to be moved and display the next unmarked message. The corresponding action is **XmhMarkMove()**.

Mark Copy Mark the selected messages to be copied into the current folder. (If the current folder is the same as the viewed folder, this command will just beep.) If no messages are highlighted, mark the current message to be copied. The corresponding action is **XmhMarkCopy()**.

Unmark Remove any of the above three marks from the selected messages, or the current message, if none are highlighted. The corresponding action is **XmhUnmark()**.

View in New Window
Create a new window containing only a view of the first selected message, or the current message, if none are highlighted. The corresponding action is **XmhViewInNewWindow()**.

Reply Create a composition window in reply to the first selected message, or the current message, if none are highlighted. The corresponding action is **XmhReply()**.

Forward Create a composition window whose body is initialized to be the contents of the selected messages, or the current message if none are highlighted. The corresponding action is **XmhForward()**.

Use as Composition Create a composition window whose body is initialized to be the contents of the first selected message, or the current message if none are selected. Any changes you make in the composition will be saved in a new message in the ''drafts'' folder, and will not change the original message. However, this command was designed to be used within the "drafts" folder to compose message drafts, and there is an exception to this rule. If the message to be used as composition was selected from the "drafts" folder, the changes will be reflected in the original message (see COMPOSITION WINDOWS). The action procedure corresponding to this command is **XmhUseAsComposition()**.

Print Print the selected messages, or the current message if none are selected. *xmh* normally prints by invoking the *enscript*(1) command, but this can be customized with the application-specific resource **PrintCommand**. The action procedure corresponding to this command is **XmhPrint()**.

SEQUENCE COMMANDS

The *Sequence* command menu contains commands pertaining to message sequences (See MESSAGE-SEQUENCES), and a list of the message-sequences defined for the currently viewed folder. The selected message-sequence is indicated by a check mark in its entry in the margin of the menu. To change the selected message-sequence, select a new message-sequence from the sequence menu.

Pick Messages Define a new message-sequence. The corresponding action is **XmhPickMessages()**.

The following menu entries will be sensitive only if the current folder has any message-sequences other than the "all" message-sequence.

Open Sequence Change the viewed sequence to be the same as the selected sequence. The corresponding action is **XmhOpenSequence()**.

Add to Sequence Add the selected messages to the selected sequence. The corresponding action is **XmhAddToSequence()**.

Remove from Sequence
Remove the selected messages from the selected sequence. The corresponding action is **XmhRemoveFromSequence()**.

Delete Sequence Remove the selected sequence entirely. The messages themselves are not affected; they simply are no longer grouped together to define a message-sequence. The corresponding action is

XmhDeleteSequence().

VIEW COMMANDS

Commands in the View menu and in the buttonboxes of view windows (which result from the Message command "View In New") correspond in functionality to commands of the same name in the Message menu, but they operate on the viewed message rather than the selected messages or current message.

Close Window	When the viewed message is in a separate view window, this command will close the view, after confirming the status of any unsaved edits. The corresponding action procedure is **XmhCloseView()**.
Reply	Create a composition window in reply to the viewed message. The related action procedure is **XmhViewReply()**.
Forward	Create a composition window whose body is initialized to be the contents of the viewed message. The corresponding action is **XmhViewForward()**.
Use As Composition	Create a composition window whose body is initialized to be the contents of the viewed message. Any changes made in the composition window will be saved in a new message in the "drafts" folder, and will not change the original message. An exception: if the viewed message was selected from the "drafts" folder, the original message is edited. The action procedure corresponding to this command is **XmhViewUseAsComposition()**.
Edit Message	This command enables the direct editing of the viewed message. The action procedure is **XmhEditView()**.
Save Message	This command is insensitive until the message has been edited; when activated, edits will be saved to the original message in the view. The corresponding action is **XmhSaveView()**.
Print	Print the viewed message. *xmh* prints by invoking the *enscript*(1) command, but this can be customized with the application-specific resource **PrintCommand**. The corresponding action procedure is **XmhPrintView()**.

OPTIONS

The *Options* menu contains one entry.

Read in Reverse
When selected, a check mark appears in the margin of this menu entry. Read in Reverse will switch the meaning of the next and previous messages, and will increment in the opposite direction. This is useful if you want to read your messages in the order of most recent first. The option acts as a toggle; select it from the menu a second time to undo the effect. The check mark appears when the option is selected.

COMPOSITION WINDOWS

Aside from the normal text editing functions, there are six command buttons associated with composition windows:

Close Window	Close this composition window. If changes have been made since the most recent Save or Send, you will be asked to confirm losing them. The corresponding action is **XmhCloseView()**.
Send	Send this composition. The corresponding action is **XmhSend()**.
New Headers	Replace the current composition with an empty message. If changes have been made since the most recent Send or Save, you will be asked to confirm losing them. The corresponding action is **XmhResetCompose()**.
Compose Message	Bring up another new composition window. The corresponding action is **XmhComposeMessage()**.
Save Message	Save this composition in your drafts folder. Then you can safely close the composition. At some future date, you can continue working on the composition by opening the drafts folder, selecting the message, and using the "Use as Composition" command. The corresponding action is **XmhSave()**.
Insert	Insert a related message into the composition. If the composition window was created with a "Reply" command, the related message is the message being replied to, otherwise no related message is defined and this button is insensitive. The message may be filtered before being inserted; see **ReplyInsertFilter**

under APPLICATION RESOURCES for more information. The corresponding action is **XmhInsert()**.

ACCELERATORS

Accelerators are shortcuts. They allow you to invoke commands without using the menus, either from the keyboard or by using the pointer.

xmh defines pointer accelerators for common actions: To select and view a message with a single click, use pointer button 2 on the message's entry in the table of contents. To select and open a folder or a sequence in a single action, make the folder or sequence selection with pointer button 2.

To mark the highlighted messages to be moved in a single action, or current message if none have been highlighted, use pointer button 3 to select the target folder. Similarly, selecting a sequence with pointer button 3 will add the highlighted or current message(s) to that sequence. In both of these operations, the selected folder or sequence and the viewed folder or sequence are not changed.

xmh defines the following keyboard accelerators over the surface of the main window, except in the view area while editing a message:

Meta-I	Incorporate New Mail
Meta-C	Commit Changes
Meta-R	Rescan Folder
Meta-P	Pack Folder
Meta-S	Sort Folder
Meta-space	View Next Message
Meta-c	Mark Copy
Meta-d	Mark Deleted
Meta-f	Forward the selected or current message
Meta-m	Mark Move
Meta-n	View Next Message
Meta-p	View Previous Message
Meta-r	Reply to the selected or current message
Meta-u	Unmark
Ctrl-V	Scroll the table of contents forward
Meta-V	Scroll the table of contents backward
Ctrl-v	Scroll the view forward
Meta-v	Scroll the view backward

TEXT EDITING COMMANDS

All of the text editing commands are actually defined by the Text widget in the Athena Widget Set. The commands may be bound to different keys than the defaults described below through the X Toolkit Intrinsics key re-binding mechanisms. See the X Toolkit Intrinsics and the Athena Widget Set documentation for more details.

Whenever you are asked to enter any text, you will be using a standard text editing interface. Various control and meta keystroke combinations are bound to a somewhat Emacs-like set of commands. In addition, the pointer buttons may be used to select a portion of text or to move the insertion point in the text. Pressing pointer button 1 causes the insertion point to move to the pointer. Double-clicking button 1 selects a word, triple-clicking selects a line, quadruple-clicking selects a paragraph, and clicking rapidly five times selects everything. Any selection may be extended in either direction by using pointer button 3.

In the following, a *line* refers to one displayed row of characters in the window. A *paragraph* refers to the text between carriage returns. Text within a paragraph is broken into lines for display based on the current width of the window. When a message is sent, text is broken into lines based upon the values of the **SendBreakWidth** and **SendWidth** application-specific resources.

The following keystroke combinations are defined:

Ctrl–a	Beginning Of Line	Meta-b	Backward Word
Ctrl–b	Backward Character	Meta-f	Forward Word
Ctrl–d	Delete Next Character	Meta-i	Insert File
Ctrl–e	End Of Line	Meta-k	Kill To End Of Paragraph
Ctrl–f	Forward Character	Meta-q	Form Paragraph
Ctrl–g	Multiply Reset	Meta-v	Previous Page
Ctrl–h	Delete Previous Character	Meta-y	Insert Current Selection
Ctrl–j	Newline And Indent	Meta-z	Scroll One Line Down
Ctrl–k	Kill To End Of Line	Meta-d	Delete Next Word
Ctrl–l	Redraw Display	Meta-D	Kill Word
Ctrl–m	Newline	Meta-h	Delete Previous Word
Ctrl–n	Next Line	Meta-H	Backward Kill Word
Ctrl–o	Newline And Backup	Meta-<	Beginning Of File
Ctrl–p	Previous Line	Meta->	End Of File
Ctrl–r	Search/Replace Backward	Meta-]	Forward Paragraph
Ctrl–s	Search/Replace Forward	Meta-[	Backward Paragraph
Ctrl–t	Transpose Characters		
Ctrl–u	Multiply by 4	Meta-Delete	Delete Previous Word
Ctrl–v	Next Page	Meta-Shift Delete	Kill Previous Word

Ctrl-w	Kill Selection	Meta-Backspace	Delete Previous Word
Ctrl-y	Unkill	Meta-Shift Backspace	Kill Previous Word
Ctrl-z	Scroll One Line Up		

In addition, the pointer may be used to cut and paste text:

Button 1 Down	Start Selection
Button 1 Motion	Adjust Selection
Button 1 Up	End Selection (cut)
Button 2 Down	Insert Current Selection (paste)
Button 3 Down	Extend Current Selection
Button 3 Motion	Adjust Selection
Button 3 Up	End Selection (cut)

CONFIRMATION DIALOG BOXES

Whenever you press a button that may cause you to lose some work or is otherwise dangerous, a popup dialog box will appear asking you to confirm the action. This window will contain an ''Abort'' or ''No'' button and a ''Confirm'' or ''Yes'' button. Pressing the ''No'' button cancels the operation, and pressing the ''Yes'' will proceed with the operation.

Some dialog boxes contain messages from *MH*. Clicking on the message field will cause the dialog box to resize so that you can read the entire message.

MESSAGE-SEQUENCES

An *MH* message sequence is just a set of messages associated with some name. They are local to a particular folder; two different folders can have sequences with the same name. In all folders, the sequence ''all'' is predefined; it consists of the set of all messages in that folder. As many as nine sequences may be defined for each folder, including the predefined ''all'' sequence. (The sequence ''cur'' is also usually defined for every folder; it consists of only the current message. *xmh* hides ''cur'' from the user, instead placing a ''+'' by the current message. Also, *xmh* does not support the ''unseen'' sequence, so that one is also hidden from the user.)

The message sequences for a folder (including one for ''all'') are displayed in the ''Sequence'' menu, below the sequence commands. The table of contents (also known as the ''toc'') is at any one time displaying one message sequence. This is called the ''viewed sequence'', and its name will be displayed in the toc title bar just after the folder name. Also, at any time one of the sequences in the menu will have a check mark next to it. This is called the ''selected sequence''. Note that the viewed sequence and the selected sequence are not necessarily the same. (This all pretty much corresponds to the way the folders work.)

The **Open Sequence, Add to Sequence, Remove from Sequence,** and **Delete Sequence** commands are active only if the viewed folder contains message-sequences.

Note that none of the above actually affect whether a message is in the folder. Remember that a sequence is a set of messages within the folder; the above operations just affect what messages are in that set.

To create a new sequence, select the ''Pick'' menu entry. A new window will appear, with lots of places to enter text. Basically, you can describe the sequence's initial set of messages based on characteristics of the message. Thus, you can define a sequence to be all the messages that were from a particular person, or with a particular subject, and so on. You can also connect things up with boolean operators, so you can select all things from ''weissman'' with the subject ''xmh''.

Hopefully, the layout is fairly obvious. The simplest cases are the easiest: just point to the proper field and type. If you enter in more than one field, it will only select messages which match all non-empty fields.

The more complicated cases arise when you want things that match one field or another one, but not necessarily both. That's what all the ''or'' buttons are for. If you want all things with the subject ''xmh'' or ''xterm'', just press the ''or'' button next to the ''Subject:'' field. Another box will appear where you can enter another subject.

If you want all things either from ''weissman'' or with subject ''xmh'', but not necessarily both, select the ''-Or-'' button. This will essentially double the size of the form. You can then enter ''weissman'' in a from: box on the top half, and ''xmh'' in a subject: box on the lower part.

If you select the ''Skip'' button, then only those messages that *don't* match the fields on that row are included.

Finally, in the bottom part of the window will appear several more boxes. One is the name of the sequence you're defining. (It defaults to the name of the selected sequence when ''Pick'' was pressed, or to ''temp'' if ''all'' was the selected sequence.) Another box defines which sequence to look through for potential members of this sequence; it defaults to the viewed sequence when ''Pick'' was pressed.

Two more boxes define a date range; only messages within that date range will be considered. These dates must be entered in 822-style format: each date is of the form ''dd mmm yy hh:mm:ss zzz'', where dd is a one or two digit day of the month, mmm is the three-letter abbreviation for a month, and yy is a year. The remaining fields are optional: hh, mm, and ss specify a time of day, and zzz selects a time zone. Note that if the time is left out, it defaults to midnight; thus if you select a range of "7 nov 86" - "8 nov 86", you will only get messages from the 7th, as all messages on the 8th will have arrived after midnight.

''Date field'' specifies which date field in the header to look at for this date range; it probably won't be useful to anyone. If the sequence you're defining already exists, you can optionally merge the old set with the new; that's what the ''Yes'' and ''No'' buttons are all about. Finally, you can ''OK'' the whole thing, or ''Cancel'' it.

In general, most people will rarely use these features. However, it's nice to occasionally use ''Pick'' to find some messages, look through them, and then hit "Delete Sequence'' to put things back in their original state.

WIDGET HIERARCHY

In order to specify resources, it is useful to know the hierarchy of widgets which compose *xmh*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name. The application class name is Xmh.

The hierarchy of the main toc and view window is identical for additional toc and view windows, except that a topLevelShell widget is inserted in the hierarchy between the application shell and the Paned widget.

```
Xmh xmh
        Paned xmh
                SimpleMenu  folderMenu
                        SmeBSB  open
                        SmeBSB  openInNew
                        SmeBSB  create
                        SmeBSB  delete
                        SmeLine  line
                        SmeBSB  close
                SimpleMenu  tocMenu
                        SmeBSB  inc
                        SmeBSB  commit
                        SmeBSB  pack
                        SmeBSB  sort
                        SmeBSB  rescan
```

```
SimpleMenu  messageMenu
        SmeBSB  compose
        SmeBSB  next
        SmeBSB  prev
        SmeBSB  delete
        SmeBSB  move
        SmeBSB  copy
        SmeBSB  unmark
        SmeBSB  viewNew
        SmeBSB  reply
        SmeBSB  forward
        SmeBSB  useAsComp
        SmeBSB  print
SimpleMenu  sequenceMenu
        SmeBSB  pick
        SmeBSB  openSeq
        SmeBSB  addToSeq
        SmeBSB  removeFromSeq
        SmeBSB  deleteSeq
        SmeLine  line
        SmeBSB  all
SimpleMenu  viewMenu
        SmeBSB  reply
        SmeBSB  forward
        SmeBSB  useAsComp
        SmeBSB  edit
        SmeBSB  save
        SmeBSB  print
SimpleMenu  optionMenu
        SmeBSB  reverse
Viewport.Core  menuBox.clip
        Box  menuBox
                MenuButton  folderButton
                MenuButton  tocButton
                MenuButton  messageButton
                MenuButton  sequenceButton
                MenuButton  viewButton
                MenuButton  optionButton
Grip  grip
Label folderTitlebar
Grip  grip
Viewport.Core  folders.clip
        Box  folders
```

```
                                    MenuButton inbox
                                    MenuButton drafts
                                            SimpleMenu menu
                                                    SmeBSB <folder_name>
                                                            .
                                                            .
                                                            .
                    Grip grip
                    Label tocTitlebar
                    Grip grip
                    Text toc
                            Scrollbar vScrollbar
                    Grip grip
                    Label viewTitlebar
                    Grip grip
                    Text view
                            Scrollbar vScrollbar
                            Scrollbar hScrollbar
```

The hierarchy of the Create Folder popup dialog box:

```
transientShell prompt
        Dialog dialog
                Label label
                Text value
                Command okay
                Command cancel
```

The hierarchy of the Notice dialog box, which reports messages from MH:

```
transientShell notice
        Dialog dialog
                Label label
                Text value
                Command confirm
```

The hierarchy of the Confirmation dialog box:

```
transientShell confirm
        Dialog dialog
                Label label
                Command yes
                Command no
```

The hierarchy of the dialog box which reports errors:

```
transientShell error
        Dialog dialog
                Label label
                Command OK
```

The hierarchy of the composition window:

```
topLevelShell xmh
        Paned xmh
                Label composeTitlebar
                Text comp
                Viewport.Core compButtons.clip
                        Box compButtons
                                Command close
                                Command send
                                Command reset
                                Command compose
                                Command save
                                Command insert
```

The hierarchy of the view window:

```
topLevelShell xmh
        Paned xmh
                Label viewTitlebar
                Text view
                Viewport.Core viewButtons.clip
                        Box viewButtons
                                Command close
                                Command reply
                                Command forward
                                Command useAsComp
                                Command edit
                                Command save
                                Command print
```

The hierarchy of the pick window:
(Unnamed widgets have no name.)

```
                    topLevelShell  xmh
                            Paned  xmh
                                    Label  pickTitlebar
                                    Viewport.core  pick.clip
                                            Form  form
                                                    Form
            The first 6 rows of the pick window have identical structure:
                                                            Form
                                                                    Toggle
                                                                    Toggle
                                                                    Label
                                                                    Text
                                                                    Command

                                                            Form
                                                                    Toggle
                                                                    Toggle
                                                                    Text
                                                                    Text
                                                                    Command
                                                            Form
                                                                    Command
                                    Viewport.core  pick.clip
                                            Form  form
                                                    From
                                                            Form
                                                                    Label
                                                                    Text
                                                                    Label
                                                                    Text
                                                            Form
                                                                    Label
                                                                    Text
                                                                    Label
                                                                    Text
                                                                    Label
                                                                    Text
                                                            Form
                                                                    Label
                                                                    Toggle
                                                                    Toggle
                                                            Form
                                                                    Command
```

Command

APPLICATION-SPECIFIC RESOURCES

Resource instance names begin with a lower case letter but are otherwise identical to the class name.

If TocGeometry, ViewGeometry, CompGeometry, or PickGeometry are not specified, then the value of Geometry is used instead. If the resulting height is not specified (e.g., "", "=500", "+0-0"), then the default height of windows is calculated from fonts and line counts. If the width is not specified (e.g., "", "=x300", "-0+0), then half of the display width is used. If unspecified, the height of a pick window defaults to half the height of the display.

Any of these options may also be specified on the command line by using the X Toolkit Intrinsics resource specification mechanism. Thus, to run *xmh* showing all message headers,

```
% xmh -xrm '*HideBoringHeaders:off'
```

The following resources are defined:

Banner A short string that is the default label of the folder, Table of Contents, and view. The default is "xmh MIT X Consortium R4".

BlockEventsOnBusy
Whether to disallow user input and show a busy cursor while *xmh* is busy processing a command. Default is true.

BusyCursor
The name of the symbol used to represent the position of the pointer, displayed if **BlockEventsOnBusy** is true, when *xmh* is processing a time-consuming command. The default is "watch".

BusyPointerColor
The foreground color of the busy cursor. Default is XtDefaultForeground.

CheckFrequency
How often to check for new mail, make checkpoints, and rescan the Table of Contents, in minutes. If **CheckNewMail** is true, *xmh* checks to see if you have new mail each interval. If **MakeCheckpoints** is true, checkpoints are made every fifth interval. Also every fifth interval, the Table of Contents is checked for inconsistencies with the file system, and rescanned. To prevent all of these checks from occurring, set **CheckFrequency** to 0. The default is 1.

CheckNewMail
: If true, *xmh* will check at regular intervals to see if new mail has arrived for any of the folders. A visual indication will be given if new mail is waiting to be retrieved. Default is True. (See BUGS). The interval can be adjusted with the **CheckFrequency**.

CommandButtonCount
: The number of command buttons to create in a button box in between the toc and the view areas of the main window. *xmh* will create these buttons with the names *button1*, *button2* and so on, in a box with the name *commandBox*. The user can specify labels and actions for the buttons in a private resource file; see the section on Actions. The default is 0.

CompGeometry
: Initial geometry for windows containing compositions.

Cursor The name of the symbol used to represent the pointer. Default is ''left_ptr''.

DraftsFolder
: The folder used for message drafts. Default is ''drafts''.

Geometry
: Default geometry to use. Default is none.

HideBoringHeaders
: If ''on'', then *xmh* will attempt to skip uninteresting header lines within messages by scrolling them off. Default is ''on''.

InitialFolder
: Which folder to display on startup. May also be set with the command-line option **-initial**. Default is ''inbox''.

InitialIncFile
: The file name of your incoming mail drop. *xmh* tries to construct a filename for the ''inc -file'' command, but in some installations (e.g. those using the Post Office Protocol) no file is appropriate. In this case, **InitialIncFile** should be specified as the empty string, and *inc* will be invoked without a -file argument. The default is to use the value of the environment variable **MAIL**, or if that is not set, to append the value of the environment variable **USER** to */usr/spool/mail/*.

MailPath
: The full path prefix for locating your mail folders. May also be set with the command-line option, **-path**. The default is the Path component in $HOME/.mh_profile, or ''$HOME/Mail'' if none.

MailWaitingFlag
If true, *xmh* will attempt to set an indication in its icon when new mail is waiting to be retrieved. If this option is true, then CheckNewMail is assumed to be true as well. The -flag command line option is a quick way to turn MailWaitingFlag on.

MakeCheckpoints
If true, *xmh* will attempt to save checkpoints of volatile information. The frequency of checkpointing is controlled by the resource **CheckFrequency**.

MhPath What directory in which to find the *MH* commands. If a command isn't found here, then the directories in the user's path are searched. Default is ''/usr/local/mh6''.

PickGeometry
Initial geometry for pick windows.

PointerColor
The foreground color of the pointer. Default is XtDefaultForeground.

PrefixWmAndIconName
Whether to prefix the window and icon name with "xmh: ". Default is true.

PrintCommand
What sh command to execute to print a message. Note that stdout and stderr must be specifically redirected! If a message or range of messages is selected for printing, the full file paths of each message file is appended to the specified print command. The default is ''enscript >/dev/null 2>/dev/null''.

ReplyInsertFilter
A shell command to be executed when the *Insert* button is activated in a composition window. The full path and filename of the source message is added to the end of the command before being passed to *sh*(1). The default filter is *cat*; i.e. it inserts the entire message into the composition. Interesting filters are: *awk -e '{print "   " $0}'* or *<mh directory>/lib/mhl -form mhl.body*.

ReverseReadOrder
When true, the next message will be the message prior to the current message in the table of contents, and the previous message will be the message after the current message in the table of contents. The default is false.

SendBreakWidth
When a message is sent from *xmh*, lines longer than this value will be split into multiple lines, each of which is no longer than **SendWidth**. This value may be overridden for a single message by inserting an additional line in the message header of the form *SendBreakWidth: value*. This line will be removed from the header before the message is sent. The default is 85.

SendWidth
When a message is sent from *xmh*, lines longer than **SendBreakWidth** characters will be split into multiple lines, each of which is no longer than this value. This value may be overridden for a single message by inserting an additional line in the message header of the form *SendWidth: value*. This line will be removed from the header before the message is sent. The default is 72.

SkipCopied
Whether to skip over messages marked for copying when using "View Next Message" and "View Previous Message". Default is true.

SkipDeleted
Whether to skip over messages marked for deletion when using "View Next Message" and "View Previous Message". Default is true.

SkipMoved
Whether to skip over messages marked for moving to other folders when using "View Next Message" and "View Previous Message". Default is true.

StickyMenu
If true, when popup command menus are used, the most recently selected entry will be under the cursor when the menu pops up. Default is false. See the file *clients/xmh/Xmh.sample* for an example of how to specify resources for pop up command menus.

TempDir
Directory for *xmh* to store temporary directories. For privacy, a user might want to change this to a private directory. Default is "/tmp".

TocGeometry
Initial geometry for master *xmh* windows.

TocPercentage
The percentage of the main window that is used to display the Table of Contents. Default is 33.

TocWidth
How many characters to generate for each message in a folder's table of contents. Default is 100. Use 80 if you plan to use *mhl* a lot, because it will be faster, and the extra 20 characters may not be useful.

ViewGeometry
Initial geometry for windows showing only a view of a message.

ACTIONS

Because *xmh* provides action procedures which correspond to command functionality and installs accelerators, users can customize accelerators in a private resource file. *xmh* provides action procedures which correspond to entries in the command menus; these are given in the sections describing menu commmands. For examples of specifying customized resources, see the file *clients/xmh/Xmh.sample*. Unpredictable results can occur if actions are bound to events or widgets for which they were not designed.

In addition to the actions corresponding to commands, these action routines are defined:

XmhPushFolder([*foldername*, ...])
This action pushes each of its argument(s) onto a stack of foldernames. If no arguments are given, the selected folder is pushed onto the stack.

XmhPopFolder() This action pops one foldername from the stack and sets the selected folder.

XmhPopupFolderMenu()
This action should always be taken when the user selects a folder button. A folder button represents a folder and zero or more subfolders. The menu of subfolders is built upon the first reference, by this routine. If there are no subfolders, this routine will mark the folder as having no subfolders, and no menu will be built. In that case the menu button emulates a toggle button. When subfolders exist, the menu will popup, using the menu button action PopupMenu().

XmhSetCurrentFolder()
: This action allows menu buttons to emulate toggle buttons in the function of selecting a folder. This action is for menu button widgets only, and sets the selected folder.

XmhLeaveFolderButton()
: This action insures that the menu button behaves properly when the user moves the pointer out of the menu button window.

XmhPushSequence([*sequencename, ...*]**)**
: This action pushes each of its arguments onto the stack of sequence names. If no arguments are given, the selected sequence is pushed onto the stack.

XmhPopSequence() This action pops one sequence name from the stack of sequence names, which then becomes the selected sequence.

XmhPromptOkayAction()
: This action is equivalent to pressing the okay button in the Create Folder popup.

XmhCancelPick() This action is equivalent to pressing the cancel button in the pick window.

CUSTOMIZATION USING *MH*

The initial text displayed in a composition window is generated by executing the corresponding *MH* command; i.e. *comp*, *repl*, or *forw*, and therefore message components may be customized as specified for those commands. *Comp* is executed only once per invocation of *xmh* and the message template is re-used for each successive new composition.

FILES

~/Mail
~/.mh_profile - *MH* profile
/usr/local/mh6 - *MH* commands
~/Mail/<folder>/.xmhcache - scan folder
~/Mail/<folder>/.mh_sequences - sequence definitions
/tmp – temporary files

SEE ALSO

X(1), xrdb(1), X Toolkit Intrinsics, Athena Widget Set, mh(1), enscript(1)

BUGS

– Printing support is minimal.
– Should handle the ''unseen'' message-sequence.
– Should determine by itself if the user hasn't used *MH* before, and offer to create the .mh_profile, instead of hanging on inc.
– Still a few commands missing (rename folder, remail message).
– A bug in *MH* limits the the number of characters in .mh_sequences to BUFSIZ. When the limit is reached, the .mh_sequences file often becomes corrupted, and sequence definitions may be lost.
– Except for the icon, there isn't an indication that you have new mail.
– There should be a resource, ShowOnInc, which when true, would show the current message in the view after incorporating new mail.
– The CheckFrequency resource should be split into two separate resources.
– WM_SAVE_YOURSELF protocol is ignored.
– WM_DELETE_WINDOW protocol doesn't work right when requesting deletion of the first toc and view, while trying to keep other *xmh* windows around.
– Doesn't support annotations when replying to messages.

COPYRIGHT

Copyright 1988, 1989, Digital Equipment Corporation.
Copyright 1989, Massachusetts Institute of Technology
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Terry Weissman, Digital Western Research Laboratory
modified by Donna Converse, MIT X Consortium

NAME

xmodmap - utility for modifying keymaps in X

SYNOPSIS

xmodmap [-options ...] [filename]

DESCRIPTION

The *xmodmap* program is used to edit and display the keyboard *modifier map* and *keymap table* that are used by client applications to convert event keycodes into keysyms. It is usually run from the user's session startup script to configure the keyboard according to personal tastes.

OPTIONS

The following options may be used with *xmodmap*:

–display *display*
: This option specifies the host and display to use.

–help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xmodmap*.

–grammar
: This option indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.

–verbose
: This option indicates that *xmodmap* should print logging information as it parses its input.

–quiet This option turns off the verbose logging. This is the default.

–n This option indicates that *xmodmap* should not change the mappings, but should display what it would do, like *make(1)* does when given this option.

–e *expression*
: This option specifies an expression to be executed. Any number of expressions may be specified from the command line.

–pm This option indicates that the current modifier map should be printed on the standard output.

–pk This option indicates that the current keymap table should be printed on the standard output.

–pp This option indicates that the current pointer map should be printed on the standard output.

– A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like *xmodmaprc*.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and parses them all before attempting execute any of them. This makes it possible to refer to keysyms that are being redefined in a natural way without having to worry as much about name conflicts.

keycode *NUMBER = KEYSYMNAME ...*
: The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal and can be determined by running the *xev* program in the examples directory). Usually only one keysym is assigned to a given code.

keysym *KEYSYMNAME = KEYSYMNAME ...*
: The *KEYSYMNAME* on the left hand side is looked up to find its current keycode and the line is replaced with the appropriate keycode expression. Note that if you have the same keysym bound to multiple keys, this might not work.

clear *MODIFIERNAME*
: This removes all entries in the modifier map for the given modifier, where valid name are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names, although it does matter for all other names). For example, "clear Lock" will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME = KEYSYMNAME ...*
: This adds the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the EXAMPLES section).

remove *MODIFIERNAME = KEYSYMNAME ...*
: This removes the given keysyms from the indicated modifier map. Unlike **add**, the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

pointer = default
: This sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = *NUMBER ...*
: This sets to pointer map to contain the indicated button codes. The list always starts with the first physical button.

Lines that begin with an exclamation point (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

Many pointers are designed such the first button is pressed using the index finger of the right hand. People who are left-handed frequently find that it is more comfortable to reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand. This could be done on a 3 button pointer as follows:

```
% xmodmap -e "pointer = 3 2 1"
```

Many editor applications support the notion of Meta keys (similar to Control keys except that Meta is held down instead of Control). However, some servers do not have a Meta keysym in the default keymap table, so one needs to be added by hand. The following command will attach Meta to the Multi-language key (sometimes label Compose Character). It also takes advantage of the fact that applications that need a Meta key simply need to get the keycode and don't require the keysym to be in the first column of the keymap table. This means that applications that are looking for a Multi_key (including the default modifier map) won't notice any change.

```
% keysym Multi_key = Multi_key Meta_L
```

One of the more simple, yet convenient, uses of *xmodmap* is to set the keyboard's "rubout" key to generate an alternate keysym. This frequently involves exchanging Backspace with Delete to be more comfortable to the user. If the *ttyModes* resource in *xterm* is set as well, all terminal emulator windows will use the same key for erasing characters:

```
% xmodmap -e "keysym BackSpace = Delete"
% echo "XTerm*ttyModes: erase ^?" | xrdb -merge
```

Some keyboards do not automatically generate less than and greater than characters when the comma and period keys are shifted. This can be remedied with *xmodmap* by resetting the bindings for the comma and period with the following scripts:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
keysym period = period greater
```

One of the more irritating differences between keyboards is the location of the Control and Shift Lock keys. A common use of *xmodmap* is to swap these two keys as follows:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

The *keycode* command is useful for assigning the same keysym to multiple keycodes. Although unportable, it also makes it possible to write scripts that can reset the keyboard to a known state. The following script sets the backspace key to generate Delete (as shown above), flushes all existing caps lock bindings, makes the CapsLock key be a control key, make F5 generate Escape, and makes Break/Reset be a shift lock.

```
!
! On the HP, the following keycodes have key caps as listed:
!
!    101  Backspace
!     55  Caps
!     14  Ctrl
!     15  Break/Reset
!     86  Stop
!     89  F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
```

```
add Lock = Caps_Lock
```

ENVIRONMENT

DISPLAY
to get default host and display number.

SEE ALSO

X(1)

BUGS

Every time a **keycode** expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium, rewritten from an earlier version by David Rosenthal of Sun Microsystems.

NAME

xpr – print an X window dump

SYNOPSIS

xpr [**–device** *dev*] [**–scale** *scale*] [**–height** *inches*] [**–width** *inches*] [**–left** *inches*] [**–top** *inches*] [**–header** *string*] [**–trailer** *string*] [**–landscape**] [**–portrait**] [**–plane** *number*] [**–gray**] [**–rv**] [**–compact**] [**–output** *filename*] [**–append** *filename*] [**–noff**] [**–split** *n*] [**–psfig**] [**–density** *dpi*] [**–cutoff** *level*] [**–noposition**] [**–gamma** *correction*] [**–render** *algorithm*] [**–slide**] [*filename*]

DESCRIPTION

xpr takes as input a window dump file produced by *xwd(1)* and formats it for output on PostScript printers, the Digital LN03 or LA100, the IBM PP3812 page printer, the HP LaserJet (or other PCL printers), or the HP PaintJet. If no file argument is given, the standard input is used. By default, *xpr* prints the largest possible representation of the window on the output page. Options allow the user to add headers and trailers, specify margins, adjust the scale and orientation, and append multiple window dumps to a single output file. Output is to standard output unless **–output** is specified.

Command Options

–device *dev*

Specifies the device on which the file will be printed. Currently supported:

la100	Digital LA100
ljet	HP LaserJet series and other monochrome PCL devices such as ThinkJet, QuietJet, RuggedWriter, HP2560 series, and HP2930 series printers
ln03	Digital LN03
pjet	HP PaintJet (color mode)
pjetxl	HP HP PaintJet XL Color Graphics Printer (color mode)
pp	IBM PP3812
ps	PostScript printer

The default device is the *LN03*, for historical reasons. **-device lw** (LaserWriter) is equivalent to -device ps and is provided only for backwards compatibility.

–scale *scale*

Affects the size of the window on the page. The PostScript, LN03, and HP printers are able to translate each bit in a window pixel map into a grid of a specified size. For example each bit might translate into a 3x3 grid. This would be specified by **–scale** *3*. By default a window is printed with the largest scale that will fit onto the page for the specified orientation.

–height *inches*
: Specifies the maximum height of the page.

–width *inches*
: Specifies the maximum width of the page.

–left *inches*
: Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

–top *inches*
: Specifies the top margin for the picture in inches. Fractions are allowed.

–header *string*
: Specifies a header string to be printed above the window.

–trailer *string*
: Specifies a trailer string to be printed below the window.

–landscape
: Forces the window to printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

–plane *number*
: Specifies which bit plane to use in an image. The default is to use the entire image and map values into black and white based on color intensities.

–gray Uses a simple 5-level gray scale conversion on a color image, rather than mapping to strictly black and white. This essentially doubles the effective width and height of the image.

–portrait
: Forces the window to be printed in portrait mode. By default a window is printed such that its longest side follows the long side of the paper.

–rv Forces the window to be printed in reverse video.

–compact
: Uses simple run-length encoding for compact representation of windows with lots of white pixels.

—output *filename*
: Specifies an output file name. If this option is not specified, standard output is used.

—append *filename*
: Specifies a filename previously produced by *xpr* to which the window is to be appended.

—noff When specified in conjunction with **–append**, the window will appear on the same page as the previous window.

—split *n* This option allows the user to split a window onto several pages. This might be necessary for very large windows that would otherwise cause the printer to overload and print the page in an obscure manner.

—psfig Suppress translation of the PostScript picture to the center of the page.

—density *dpi*
: Indicates what dot-per-inch density should be used by the HP printer.

—cutoff *level*
: Changes the intensity level where colors are mapped to either black or white for monochrome output on a LaserJet printer. The *level* is expressed as percentage of full brightness. Fractions are allowed.

—noposition
: This option causes header, trailer, and image positioning command generation to be bypassed for LaserJet, PaintJet and PaintJet XL printers.

—gamma *correction*
: This changes the intensity of the colors printed by PaintJet XL printer. The *correction* is a floating point value in the range 0.00 to 3.00. Consult the operator's manual to determine the correct value for the specific printer.

—render *algorithm*
: This allows PaintJet XL printer to render the image with the best quality versus performance tradeoff. Consult the operator's manual to determine which *algorithms* are available.

—slide This option allows overhead transparencies to be printed using the PaintJet and PaintJet XL printers.

SEE ALSO

xwd(1), xwud(1), X(1)

LIMITATIONS

The current version of *xpr* can generally print out on the LN03 most X windows that are not larger than two-thirds of the screen. For example, it will be able to print out a large Emacs window, but it will usually fail when trying to print out the entire screen. The LN03 has memory limitations that can cause it to incorrectly print very large or complex windows. The two most common errors encountered are ''band too complex'' and ''page memory exceeded.'' In the first case, a window may have a particular six pixel row that contains too many changes (from black to white to black). This will cause the printer to drop part of the line and possibly parts of the rest of the page. The printer will flash the number '1' on its front panel when this problem occurs. A possible solution to this problem is to increase the scale of the picture, or to split the picture onto two or more pages. The second problem, ''page memory exceeded,'' will occur if the picture contains too much black, or if the picture contains complex half-tones such as the background color of a display. When this problem occurs the printer will automatically split the picture into two or more pages. It may flash the number '5' on its from panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or to rework the application to produce a less complex picture.

There are several limitations on the LA100 support: the picture will always be printed in portrait mode, there is no scaling, and the aspect ratio will be slightly off.

Support for PostScript output currently cannot handle the **-append**, **-noff** or **-split** options.

The **-compact** option is *only* supported for PostScript output. It compresses white space but not black space, so it is not useful for reverse-video windows.

For color images, should map directly to PostScript image support.

HP PRINTERS

If no —**density** is specified on the command line 300 dots per inch will be assumed for *ljet* and 90 dots per inch for *pjet*. Allowable *density* values for a LaserJet printer are 300, 150, 100, and 75 dots per inch. Consult the operator's manual to determine densities supported by other printers.

If no —**scale** is specified the image will be expanded to fit the printable page area.

The default printable page area is 8x10.5 inches. Other paper sizes can be accomodated using the **–height** and **–width** options.

Note that a 1024x768 image fits the default printable area when processed at 100 dpi with scale=1, the same image can also be printed using 300 dpi with scale=3 but will require considerably more data be transfered to the printer.

xpr may be tailored for use with monochrome PCL printers other than the LaserJet. To print on a ThinkJet (HP2225A) *xpr* could be invoked as:

xpr -density 96 -width 6.667 *filename*

or for black-and-white output to a PaintJet:

xpr -density 180 *filename*

The monochrome intensity of a pixel is computed as 0.30*R + 0.59*G + 0.11*B. If a pixel's computed intensity is less than the **–cutoff** level it will print as white. This maps light-on-dark display images to black-on-white hardcopy. The default cutoff intensity is 50% of full brightness. Example: specifying **–cutoff 87.5** moves the white/black intensity point to 87.5% of full brightness.

A LaserJet printer must be configured with sufficient memory to handle the image. For a full page at 300 dots per inch approximately 2MB of printer memory is required.

Color images are produced on the PaintJet at 90 dots per inch. The PaintJet is limited to sixteen colors from its 330 color palette on each horizontal print line. *xpr* will issue a warning message if more than sixteen colors are encountered on a line. *xpr* will program the PaintJet for the first sixteen colors encountered on each line and use the nearest matching programmed value for other colors present on the line.

Specifying the **–rv**, reverse video, option for the PaintJet will cause black and white to be interchanged on the output image. No other colors are changed.

Multiplane images must be recorded by *xwd* in *ZPixmap* format. Single plane (monochrome) images may be in either *XYPixmap* or *ZPixmap* format.

Some PCL printers do not recognize image positioning commands. Output for these printers will not be centered on the page and header and trailer strings may not appear where expected.

The —**gamma** and -**render** options are supported only on the PaintJet XL printers.

The —**slide** option is not supported for LaserJet printers.

The —**split** option is not supported for HP printers.

COPYRIGHT

AUTHORS

Michael R. Gretzinger, MIT Project Athena, Jose Capo, MIT Project Athena (PP3812 support), Marvin Solomon, University of Wisconsin, Bob Scheifler, MIT, Angela Bock and E. Mike Durbin, Rich Inc. (grayscale), Larry Rupp, HP (HP printer support).

NAME

xprop - property displayer for X

SYNOPSIS

xprop [-help] [-grammar] [-id *id*] [-root] [-name *name*] [-frame] [-font *font*] [-display *display*] [-len *n*] [-notype] [-fs *file*] [-f *atom format* [*dformat*]]* [*format* [*dformat*] *atom*]*

SUMMARY

The *prop* utility is for displaying window and font properties in an X server. One window or font is selected using the command line arguments or possibly in the case of a window, by clicking on the desired window. A list of properties is then given, possibly with formatting information.

OPTIONS

-help Print out a summary of command line options.

-grammar
Print out a detailed grammar for all command line options.

-id *id* This argument allows the user to select window *id* on the command line rather than using the pointer to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the pointer might be impossible or interfere with the application.

-name *name*
This argument allows the user to specify that the window named *name* is the target window on the command line rather than using the pointer to select the target window.

-font *font*
This argument allows the user to specify that the properties of font *font* should be displayed.

-root This argument specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

-display *display*
This argument allows you to specify the server to connect to; see *X(1)*.

-len *n* Specifies that at most *n* bytes of any property should be read or displayed.

-notype Specifies that the type of each property should not be displayed.

-fs *file* Specifies that file *file* should be used as a source of more formats for properties.

-frame Specifies that when selecting a window by hand (i.e. if none of **-name**, **-root**, or **-id** are given), look at the window manager frame (if any) instead of looking for the client window.

-remove *property-name*
Specifies the name of a property to be removed from the indicated window.

-f *name format* [*dformat*]
Specifies that the *format* for *name* should be *format* and that the *dformat* for *name* should be *dformat*. If *dformat* is missing, " = $0+\n" is assumed.

DESCRIPTION

For each of these properties, its value on the selected window or font is printed using the supplied formatting information if any. If no formatting information is supplied, internal defaults are used. If a property is not defined on the selected window or font, "not defined" is printed as the value for that property. If no property list is given, all the properties possessed by the selected window or font are printed.

A window may be selected in one of four ways. First, if the desired window is the root window, the -root argument may be used. If the desired window is not the root window, it may be selected in two ways on the command line, either by id number such as might be obtained from *xwininfo*, or by name if the window possesses a name. The -id argument selects a window by id number in either decimal or hex (must start with 0x) while the -name argument selects a window by name.

The last way to select a window does not involve the command line at all. If none of -font, -id, -name, and -root are specified, a crosshairs cursor is displayed and the user is allowed to choose any visible window by pressing any pointer button in the desired window. If it is desired to display properties of a font as opposed to a window, the -font argument must be used.

Other than the above four arguments and the -help argument for obtaining help, and the -grammar argument for listing the full grammar for the command line, all the other command line arguments are used in specifing both the format of the properties to be displayed and how to display them. The -len *n* argument specifies that at most *n* bytes of any given property will be read and displayed. This is useful for example when displaying the cut buffer on the root window which could run to several pages if displayed in full.

Normally each property name is displayed by printing first the property name then its type (if it has one) in parentheses followed by its value. The -notype argument specifies that property types should not be displayed. The -fs argument is used to specify a file containing a list of formats for properties while the -f argument is used to specify the format for one property.

The formatting information for a property actually consists of two parts, a *format* and a *dformat*. The *format* specifies the actual formatting of the property (i.e., is it made up of words, bytes, or longs?, etc.) while the *dformat* specifies how the property should be displayed.

The following paragraphs describe how to construct *formats* and *dformats*. However, for the vast majority of users and uses, this should not be necessary as the built in defaults contain the *formats* and *dformats* necessary to display all the standard properties. It should only be necessary to specify *formats* and *dformats* if a new property is being dealt with or the user dislikes the standard display format. New users especially are encouraged to skip this part.

A *format* consists of one of 0, 8, 16, or 32 followed by a sequence of one or more format characters. The 0, 8, 16, or 32 specifies how many bits per field there are in the property. Zero is a special case meaning use the field size information associated with the property itself. (This is only needed for special cases like type INTEGER which is actually three different types depending on the size of the fields of the property)

A value of 8 means that the property is a sequence of bytes while a value of 16 would mean that the property is a sequence of words. The difference between these two lies in the fact that the sequence of words will be byte swapped while the sequence of bytes will not be when read by a machine of the opposite byte order of the machine that orginally wrote the property. For more information on how properties are formatted and stored, consult the Xlib manual.

Once the size of the fields has been specified, it is necessary to specify the type of each field (i.e., is it an integer, a string, an atom, or what?) This is done using one format character per field. If there are more fields in the property than format characters supplied, the last character will be repeated as many times as necessary for the extra fields. The format characters and their meaning are as follows:

a	The field holds an atom number. A field of this type should be of size 32.
b	The field is an boolean. A 0 means false while anything else means true.

c	The field is an unsigned number, a cardinal.
i	The field is a signed integer.
m	The field is a set of bit flags, 1 meaning on.
s	This field and the next ones until either a 0 or the end of the property represent a sequence of bytes. This format character is only usable with a field size of 8 and is most often used to represent a string.
x	The field is a hex number (like 'c' but displayed in hex - most useful for displaying window ids and the like)

An example *format* is 32ica which is the format for a property of three fields of 32 bits each, the first holding a signed integer, the second an unsigned integer, and the third an atom.

The format of a *dformat* unlike that of a *format* is not so rigid. The only limitations on a *dformat* is that one may not start with a letter or a dash. This is so that it can be distingished from a property name or an argument. A *dformat* is a text string containing special characters instructing that various fields be printed at various points in a manner similar to the formatting string used by printf. For example, the *dformat* " is ($0, $1 \)\n" would render the POINT 3, -4 which has a *format* of 32ii as " is (3, -4)\n".

Any character other than a $, ?, \, or a (in a *dformat* prints as itself. To print out one of $, ?, \, or (preceed it by a \. For example, to print out a $, use \$. Several special backslash sequences are provided as shortcuts. \n will cause a newline to be displayed while \t will cause a tab to be displayed. \o where *o* is an octal number will display character number *o*.

A $ followed by a number *n* causes field number *n* to be displayed. The format of the displayed field depends on the formatting character used to describe it in the corresponding *format*. I.e., if a cardinal is described by 'c' it will print in decimal while if it is described by a 'x' it is displayed in hex.

If the field is not present in the property (this is possible with some properties), <field not available> is displayed instead. $*n*+ will display field number *n* then a comma then field number *n*+1 then another comma then ... until the last field defined. If field *n* is not defined, nothing is displayed. This is useful for a property that is a list of values.

A ? is used to start a conditional expression, a kind of if-then statement. ?*exp*(*text*) will display *text* if and only if *exp* evaluates to non-zero. This is useful for two things. First, it allows fields to be displayed if and only if a flag is set. And second, it allows a value such as a state number to be displayed as a name rather than as just a number. The syntax of *exp* is as follows:

exp	::= *term* | *term=exp* | !*exp*
term	::= *n* | $*n* | m*n*

The ! operator is a logical "not", changing 0 to 1 and any non-zero value to 0. = is an equality operator. Note that internally all expressions are evaluated as 32 bit numbers so -1 is not equal to 65535. = returns 1 if the two values are equal and 0 if not. *n* represents the constant value *n* while $*n* represents the value of field number *n*. m*n* is 1 if flag number *n* in the first field having format character 'm' in the corrsponding *format* is 1, 0 otherwise.

Examples: ?m3(count: $3\n) displays field 3 with a label of count if and only if flag number 3 (count starts at 0!) is on. ?$2=0(True)?!$2=0(False) displays the inverted value of field 2 as a boolean.

In order to display a property, *xprop* needs both a *format* and a *dformat*. Before *xprop* uses its default values of a *format* of 32x and a *dformat* of " = { $0+ }\n", it searches several places in an attempt to find more specific formats. First, a search is made using the name of the property. If this fails, a search is made using the type of the property. This allows type STRING to be defined with one set of formats while allowing property WM_NAME which is of type STRING to be defined with a different format. In this way, the display formats for a given type can be overridden for specific properties.

The locations searched are in order: the format if any specified with the property name (as in 8x WM_NAME), the formats defined by -f options in last to first order, the contents of the file specified by the -fs option if any, the contents of the file specified by the environmental variable XPROPFORMATS if any, and finally *xprop*'s built in file of formats.

The format of the files refered to by the -fs argument and the XPROPFORMATS variable is one or more lines of the following form:

name format [*dformat*]

Where *name* is either the name of a property or the name of a type, *format* is the *format* to be used with *name* and *dformat* is the *dformat* to be used with *name*. If *dformat* is not present, " = $0+\n" is assumed.

EXAMPLES

To display the name of the root window: *xprop* -root WM_NAME

To display the window manager hints for the clock: *xprop* -name xclock WM_HINTS

To display the start of the cut buffer: *xprop* -root -len 100 CUT_BUFFER0

To display the point size of the fixed font: *xprop* -font fixed POINT_SIZE

To display all the properties of window # 0x200007: *xprop* -id 0x200007

ENVIRONMENT

DISPLAY
To get default display.

XPROPFORMATS
Specifies the name of a file from which additional formats are to be obtained.

SEE ALSO

X(1), xwininfo(1)

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xpseudoroot – create a pseudo root window

SYNOPSIS

xpseudoroot [-options ...] *property_name*

DESCRIPTION

The *xpseudoroot* program allows you to create pseudo root windows as outlined in the Inter-Client Communications Conventions Manual. By default it just makes a copy of the normal root window, but command line options may be used to alter much of the screen-related information.

The command line argument *property_name* specifies the name of a property on the screen's real root window in which to store the pseudo root information. Applications can be run within the pseudo root window by appending *.property_name* to the *displaynumber.screennumber* part of the display name; for example: expo:0.0.*property_name*

WARNING: This is experimental code for implementing pseudo root windows as specified by the Inter-Client Communications Conventions Manual. The interfaces that it provides should be considered private to the MIT implementation of Xlib and WILL CHANGE IN THE NEXT RELEASE. The interfaces that it provides should not be incorporated into any toolkits or applications. No effort will be made to provide backward compatibility.

OPTIONS

-display *displayname*
This option specifies the name of the X server to contact.

-geometry *geom*
This option specifies size and location of the pseudo root window.

-visuals *visualid ...*
This option specifies a list of visuals to support on the pseudo root window. Any number of numeric visual identifiers (in hex, octal, or decimal) may be given per *-visuals.*

-colormap *colormapid*
This option specifies the numeric colormap identifier to be associated with the pseudo root window.

-Colormap *visualid*
This option specifies a numeric visual identifier to be used in creating a new colormap for the pseudo root window. If this option is given, *xpseudoroot* will create a new colormap from the given visual and set the black and white pixel fields to the desired colors.

-white *pixel*
This option specifies the numeric pixel value to use for WhitePixel when creating a new colormap with *–Colormap.* The default is to copy the real screen's WhitePixel.

-White *colorname*
This option specifies the color to use when setting WhitePixel in newly created colormaps. It may be used with *–white* to create arbitrary WhitePixels.

-black *pixel*
This option specifies the numeric pixel value to use for BlackPixel when creating a new colormap with *–Colormap.* The default is to copy the real screen's BlackPixel.

-Black *colorname*
This option specifies the color to use when setting BlackPixel in newly created colormaps. It may be used with *–black* to create arbitrary BlackPixels.

-empty This option indicates that any colormaps created with *–Colormap* should not have BlackPixel and WhitePixel preallocated (although the values may still be set with *–black* and *–white*). This leaves as much room as possible for running applications that would otherwise not find enough colors. This is not for general use as it guarantees that an application will be displayed in incorrect colors.

-max *number*
This option specifies the maximum number of installed colormaps that will be allowed on this screen. The default is to use the real screen's value.

-min *number*
This option specifies the minimum number of installed colormaps that will be allowed on this screen. The default is to use the real screen's value.

-backingstore *when*
This option specifies when backing store window attributes will be honored and takes one of the following arguments: **NotUseful**, **WhenMapped**, or **Always**. The default is to use the real screen's value.

-saveunders *boolean*
This option specifies whether or not this screen supports saveunders and takes one of the following arguments: *yes* or *no.*

-name *string*
This option specifies the name to be used for the pseudo root window.

SEE ALSO

X(1), xdpyinfo(1), xwininfo(1), xprop(1), Inter-Client Communications Conventions Manual

BUGS

This is a sample program that is primarily intended as a testbed for ICCCM pseudo roots. It should not be incorporated into any toolkit or application. Both the command line arguments and internal interfaces are **guaranteed** to change in the next release.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xrdb — X server resource database utility

SYNOPSIS

xrdb [-option ...] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file.

The resource manager (used by the Xlib routine *XGetDefault(3X)* and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editting files.

For compatibility, if there is no RESOURCE_MANAGER property defined (either because xrdb was not run or if the property was removed), the resource manager will look for a file called *.Xdefaults* in your home directory.

The *filename* (or the standard input if - or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

HOST=hostname
: the hostname portion of the display to which you are connected.

WIDTH=num
: the width of the screen in pixels.

HEIGHT=num
: the height of the screen in pixels.

X_RESOLUTION=num
: the x resolution of the screen in pixels per meter.

Y_RESOLUTION=num
: the y resolution of the screen in pixels per meter.

PLANES=num
: the number of bit planes for the default visual.

BITS_PER_RGB=num
: the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it is not related to the number of planes, which the log base 2 of the size of the

NAME

xrdb - X server resource database utility

SYNOPSIS

xrdb [-option ...] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file.

The resource manager (used by the Xlib routine *XGetDefault(3X)* and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editting files.

For compatibility, if there is no RESOURCE_MANAGER property defined (either because xrdb was not run or if the property was removed), the resource manager will look for a file called *Xdefaults* in your home directory.

The *filename* (or the standard input if - or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

BITS_PER_RGB=num
: the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it usually is not related to PLANES.

CLASS=visualclass
: one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor. This is the visual class of the root window of the default screen.

COLOR defined only if CLASS is one of StaticColor, PseudoColor, TrueColor, or DirectColor.

HEIGHT=num
: the height of the default screen in pixels.

HOST=hostname
: the hostname portion of the display to which you are connected.

PLANES=num
the number of bit planes (the depth) of the root window of the default screen.

RELEASE=num
the vendor release number for the server. The interpretation of this number will vary depending on VENDOR.

REVISION=num
the X protocol minor version supported by this server (currently 0).

VERSION=num
the X protocol major version supported by this server (should always be 11).

VENDOR=vendor
a string specifying the vendor of the server.

WIDTH=num
the width of the default screen in pixels.

X_RESOLUTION=num
the x resolution of the default screen in pixels per meter.

Y_RESOLUTION=num
the y resolution of the default screen in pixels per meter.

Lines that begin with an exclamation mark (!) are ignored and may be used as comments.

OPTIONS

xrdb program accepts the following options:

–help This option (or any unsupported option) will cause a brief description of the allowable options and parameters to be printed.

–display *display*
This option specifies the X server to be used; see *X(1)*.

–n This option indicates that changes to the property (when used with *-load*) or to the resource file (when used with *-edit*) should be shown on the standard output, but should not be performed.

–quiet This option indicates that warning about duplicate entries should not be displayed.

-cpp *filename*
This option specifies the pathname of the C preprocessor program to be used. Although *xrdb* was designed to use CPP, any program that acts as a filter and accepts the -D, -I, and -U options may be used.

-nocpp This option indicates that *xrdb* should not run the input file through a preprocessor before loading it into the RESOURCE_MANAGER property.

–symbols
This option indicates that the symbols that are defined for the preprocessor should be printed onto the standard output. It can be used in conjunction with **–query**, but not with the options that change the RESOURCE_MANAGER property.

–query This option indicates that the current contents of the RESOURCE_MANAGER property should be printed onto the standard output. Note that since preprocessor commands in the input resource file are part of the input file, not part of the property, they won't appear in the output from this option. The –edit option can be used to merge the contents of the property back into the input resource file without damaging preprocessor commands.

–load This option indicates that the input should be loaded as the new value of the RESOURCE_MANAGER property, replacing whatever was there (i.e. the old contents are removed). This is the default action.

–merge This option indicates that the input should be merged with, instead of replacing, the current contents of the RESOURCE_MANAGER property. Since *xrdb* can read the standard input, this option can be used to the change the contents of the RESOURCE_MANAGER property directly from a terminal or from a shell script.

–remove
This option indicates that the RESOURCE_MANAGER property should be removed from its window.

–retain This option indicates that the server should be instructed not to reset of *xrdb* is the first client.

–edit *filename*
This option indicates that the contents of the RESOURCE_MANAGER property should be edited into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

−backup *string*
This option specifies a suffix to be appended to the filename used with **−edit** to generate a backup file.

−D*name[=value]*
This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef*.

−U*name* This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

−I*directory*
This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include*.

FILES

Generalizes *~/.Xdefaults* files.

SEE ALSO

X(1), XGetDefault(3X), Xlib Resource Manager documentation

ENVIRONMENT

DISPLAY
to figure out which display to use.

BUGS

The default for no arguments should be to query, not to overwrite, so that it is consistent with other programs.

COPYRIGHT

Copyright 1988, Digital Equipment Corporation.

AUTHORS

Phil Karlton, rewritten from the original by Jim Gettys

NAME

xrefresh - refresh all or part of an X screen

SYNOPSIS

xrefresh [-option ...]

DESCRIPTION

Xrefresh is a simple X program that causes all or part of your screen to be repainted. This is useful when system messages have messed up your screen. *Xrefresh* maps a window on top of the desired area of the screen and then immediately unmaps it, causing refresh events to be sent to all applications. By default, a window with no background is used, causing all applications to repaint "smoothly." However, the various options can be used to indicate that a solid background (of any color) or the root window background should be used instead.

ARGUMENTS

–white Use a white background. The screen just appears to flash quickly, and then repaint.

–black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.

–solid *color*
Use a solid background of the specified color. Try green.

–root Use the root window background.

–none This is the default. All of the windows simply repaint.

–geometry *WxH+X+Y*
Specifies the portion of the screen to be repainted; see *X(1)*.

–display *display*
This argument allows you to specify the server and screen to refresh; see *X(1)*.

X DEFAULTS

The *xrefresh* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

Black, White, Solid, None, Root
Determines what sort of window background to use.

Geometry
Determines the area to refresh. Not very useful.

ENVIRONMENT

DISPLAY - To get default host and display number.

SEE ALSO

X(1)

BUGS

It should have just one default type for the background.

COPYRIGHT

AUTHORS

Jim Gettys, Digital Equipment Corp., MIT Project Athena

NAME

xscope - X Window Protocol Viewer

SYNOPSIS

xscope [option] ...

DESCRIPTION

Xscope sits in-between an X11 client and an X11 server and prints the contents of each request, reply, error, or event that is communicated between them. This information can be useful in debugging and performance tuning of X11 servers and clients.

To operate, *xscope* must know the host, port, and display to use to connect to the X11 server. In addition, it must know the port on which it should listen for X11 clients. Two cases are common:

(1) The X11 server is on the same host as *xscope*.
In this case, the input port for *xscope* should be selected as an X11 server on a different display, and the client DISPLAY argument adjusted to select *xscope* . For example, if the X11 server is on port 6000, display 1, then *xscope* can use port 6002 as its input port. The client can use display 1 for direct access to X11 or display 2 for access to *xscope*.

(2) The X11 server is on a different host than *xscope*.
In this case the same input and output ports can be used, and the host component of the DISPLAY is used to select *xscope* or X11.

ARGUMENTS

—i<input-port>
Specify the port that *xscope* will use to take requests from clients (defaults to 1). For X11, this port is automatically biased by 6000.

—o<output-port>
Determines the port that *xscope* will use to connect to X11 (defaults to 0). For X11, this port is automatically biased by 6000.

—h<host> Determines the host that *xscope* will use to find its X11 server.

—d<display>
Defines the display number. The display number is added to the input and output port to give the actual ports which are used by *xscope*.

—q Quiet output mode. Gives only the names of requests, replies, errors, and events, but does not indicate contents.

−v<print-level>

Determines the level of printing which *xscope* will provide. The print-level can be 0 (same as quiet mode), 1, 2, 3, 4. The larger numbers give more and more output. For example, a successful setup returns a string which is the name of the vendor of the X11 server. At level 1, the explicit field giving the length of the string is suppressed since it can be inferred from the string. At level 2 and above the length is explicitly printed.

EXAMPLES

```
xscope -v4 -hcleo -d0 -o0 -i1
```

This command would have xscope communicate with an X11 server on host ''cleo'', display 0; xscope itself would be available on the current host as display 1 (display of 0 plus the 1 of -i1). Verbose level 4.

```
xscope -q -d1 -o1 -o3
```

The X11 server for the current host, display 2 (1 for -d1 plus 1 for -o1) would be used by xscope which would run as display 4 (1 for -d1 plus 3 for -o3). Quite mode (verbose level 0).

SEE ALSO

X(1), X11 Protocol document (doc/Protocol/X11.protocol)

AUTHOR

James L. Peterson (MCC)

Copyright 1988, MCC

BUGS

Code has only been tested on Sun3's.

NAME

X - X Window System server

SYNOPSIS

X [:displaynumber] [-option ...] [ttyname]

DESCRIPTION

X is the generic name for the X Window System server. It is frequently a link or a copy of the appropriate server binary for driving the most frequently used server on a given machine. The sample server from MIT supports the following platforms:

Xqvss	Digital monochrome vaxstationII or II
Xqdss	Digital color vaxstationII or II
Xsun	Sun monochrome or color Sun 2, 3, or 4
Xhp	HP Topcat 9000s300
Xapollo	Apollo monochrome (Domain/IX 9.6)
Xibm	IBM APA and megapel PC/RT
XmacII	Apple monochrome Macintosh II
Xplx	Parallax color and video graphics controller

STARTING THE SERVER

The server is usually started from the X Display Manager program *xdm*. This utility is run from the system boot files and takes care of keeping the server running, prompting for usernames and passwords, and starting up the user sessions. It is easily configured for sites that wish to provide nice, consistent interfaces for novice users (loading convenient sets of resources, starting up a window manager, clock, and nice selection of terminal emulator windows).

Since *xdm* now handles automatic starting of the server in a portable way, the *-L* option to *xterm* is now considered obsolete. Support for starting a login window from 4.3bsd-derived */etc/ttys* files may not be included in future releases.

Installations that run more than one window system will still need to use the *xinit* utility. However, *xinit* is to be considered a tool for building startup scripts and is not intended for use by end users. Site adminstrators are strongly urged to build nicer interfaces for novice users.

When the sample server starts up, it takes over the display. If you are running on a workstation whose console is the display, you cannot log into the console while the server is running.

NETWORK CONNECTIONS

The sample server supports connections made using the following reliable byte-streams:

TCP/IP
: The server listens on port htons(6000+*n*), where *n* is the display number.

Unix Domain
: The sample server uses */tmp/X11-unix/X***n** as the filename for the socket, where *n* is the display number.

DECnet
: The server responds to connections to object *X$X***n**, where *n* is the display number.

OPTIONS

All of the sample servers accept the following command line options:

—a *number*
: sets pointer acceleration (i.e. the ratio of how much is reported to how much the user actually moved the pointer).

—auth *authorization-file*
: Specifies a file which contains a collection of authorization records used to authenticate access.

bc disables certain kinds of error checking, for bug compatibility with previous releases (e.g., to work around bugs in R2 and R3 xterms and toolkits). Deprecated.

—bs disables backing store support on all screens.

—c turns off key-click.

c *volume* sets key-click volume (allowable range: 0-100).

-cc *class* sets the visual class for the root window of color screens. The class numbers are as specified in the X protocol. Not obeyed by all servers.

—co *filename*
: sets name of RGB color database.

—f *volume*
: sets feep (bell) volume (allowable range: 0-100).

—fc *cursorFont*
: sets default cursor font.

—fn *font* sets the default font.

—fp *fontPath*
: sets the search path for fonts.

—help	prints a usage message.
—I	causes all remaining command line arguments to be ignored.
—ld *kilobytes*	sets the data space limit of the server to the specified number of kilobytes. The default value is zero, making the data size as large as possible. A value of -1 leaves the data space limit unchanged. This option is not available in all operating systems.
—ls *kilobytes*	sets the stack space limit of the server to the specified number of kilobytes. The default value is zero, making the stack size as large as possible. A value of -1 leaves the stack space limit unchanged. This option is not available in all operating systems.
—logo	turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client.
nologo	turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.
—p *minutes*	sets screen-saver pattern cycle time in minutes.
—r	turns off auto-repeat.
r	turns on auto-repeat.
—s *minutes*	sets screen-saver timeout time in minutes.
—su	disables save under support on all screens.
—t *numbers*	sets pointer acceleration threshold in pixels (i.e. after how many pixels pointer acceleration should take effect).
—to *seconds*	sets default connection timeout in seconds.
tty*xx*	ignored, for servers started the ancient way (from init).
v	sets video-on screen-saver preference.
—v	sets video-off screen-saver preference.
—wm	forces the default backing-store of all windows to be When-Mapped; a cheap trick way of getting backing-store to apply to all windows.

–x *extension*
: loads the specified extension at init. Not supported in most implementations.

Many servers also have device-specific command line options. See the manual pages for the individual servers for more details.

SECURITY

The sample server implements a simplistic authorization protocol which uses data private to authorized clients and the server. The authorzation data is passed to the server in a private file named with the **-auth** command line option. If this file contains any authorization records, the local host is not automatically allowed access to the server, and only clients which send one of the authorization records contained in the file in the connection setup information will be allowed access. The authorization name supported is "MIT-MAGIC-COOKIE-1". See the *Xau* manual page for a description of the binary format of this file. Maintenence of this file, and distribution of its contents to remote sites for use there is left as an exercise for the reader.

The sample server also uses a host-based access control list for deciding whether or not to accept connections from clients on a particular machine. This list initially consists of the host on which the server is running as well as any machines listed in the file */etc/X***n***.hosts*, where **n** is the display number of the server. Each line of the file should contain either an Internet hostname (e.g. expo.lcs.mit.edu) or a DECnet hostname in double colon format (e.g. hydra::). There should be no leading or trailing spaces on any lines. For example:

```
joesworkstation
corporate.company.com
star::
bigcpu::
```

Users can add or remove hosts from this list and enable or disable access control using the *xhost* command from the same machine as the server. For example:

```
% xhost +janesworkstation
janesworkstation being added to access control list
% xhost -star::
public:: being removed from access control list
% xhost +
all hosts being allowed (access control disabled)
% xhost -
all hosts being restricted (access control enabled)
% xhost
```

```
access control enabled (only the following hosts are allowed)
joesworkstation
janesworkstation
corporate.company.com
bigcpu::
```

Unlike some window systems, X does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen. Sites that have better authentication and authorization systems (such as Kerberos) might wish to make use of the hooks in the libraries and the server to provide additional security models.

SIGNALS

The sample server attaches special meaning to the following signals:

SIGHUP This signal causes the server to close all existing connections, free all resources, and restore all defaults. It is sent by the display manager whenever the main user's main application (usually an *xterm* or window manager) exits to force the server to clean up and prepare for the next user.

SIGTERM
This signal causes the server to exit cleanly.

FONTS

Fonts are usually stored as individual files in directories. The list of directories in which the server looks when trying to open a font is controlled by the *font path.* Although most sites will choose to have the server start up with the appropriate font path (using the *-fp* option mentioned above), it can be overridden using the *xset* program.

The default font path for the sample server contains three directories:

/usr/lib/X11/fonts/misc
This directory contains several miscellaneous fonts that are useful on all systems. It contains a very small family of fixed-width fonts (**6x10**, **6x12**, **6x13**, **8x13**, **8x13bold**, and **9x15**) and the cursor font. It also has font name aliases for the commonly used fonts **fixed** and **variable**.

/usr/lib/X11/fonts/75dpi
This directory contains fonts contributed by Adobe Systems, Inc. and Digital Equipment Corporation and by Bitstream, Inc. for 75 dots per inch displays. An integrated selection of sizes, styles, and weights are provided for each family.

/usr/lib/X11/fonts/100dpi
This directory contains versions of some of the fonts in the *75dpi* directory for 100 dots per inch displays.

Font databases are created by running the *mkfontdir* program in the directory containing the compiled versions of the fonts (the *.snf* files). Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. **If *mkfontdir* is not run, the server will not be able to find any fonts in the directory.**

DIAGNOSTICS

Too numerous to list them all. If run from *init(8)*, errors are logged in the file */usr/adm/X*msgs*,

FILES

/etc/X*.hosts	Initial access control list
/usr/lib/X11/fonts/misc, /usr/lib/X11/fonts/75dpi, /usr/lib/X11/fonts/100dpi	Font directories
/usr/lib/X11/rgb.txt	Color database
/tmp/.X11-unix/X*	Unix domain socket
/usr/adm/X*msgs	Error log file

SEE ALSO

X(1), xdm(1), mkfontdir(1), xinit(1), xterm(1), twm(1), xhost(1), xset(1), xsetroot(1), ttys(5), init(8), Xqdss(1), Xqvss(1), Xsun(1), Xapollo(1), XmacII(1) *X Window System Protocol, Definition of the Porting Layer for the X v11 Sample Server, Strategies for Porting the X v11 Sample Server, Godzilla's Guide to Porting the X V11 Sample Server*

BUGS

The option syntax is inconsistent with itself and *xset(1)*.

The acceleration option should take a numerator and a denominator like the protocol.

If *X* dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The color database is missing a large number of colors. However, there doesn't seem to be a better one available that can generate RGB values tailorable to particular displays.

The *xterm -L* method for starting an initial window from */etc/ttys* is completely inadequate and should be removed. People should use *xdm* instead.

COPYRIGHT

Copyright 1984, 1985, 1986, 1987, 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHORS

The sample server was originally written by Susan Angebranndt, Raymond Drewry, Philip Karlton, and Todd Newman, with support from a cast of thouands. See also the file *doc/contributors* in the sample distribution for a more complete list.

NAME

xset - user preference utility for X

SYNOPSIS

xset [-display *display*] [-b] [b on/off] [b [*volume* [*pitch* [*duration*]]] [-c] [c on/off] [c [*volume*]] [[-+]fp[-+=] *path*[,*path*[,...]]] [fp default] [fp rehash] [[-]led [*integer*]] [led on/off] [m[ouse] [*acceleration* [*threshold*]]] [m[ouse] default] [p *pixel color*] [[-]r] [r on/off] [s [*length* [*period*]]] [s blank/noblank] [s expose/noexpose] [s on/off] [s default] [q]

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS

−display *display*

This option specifies the server to use; see *X(1)*.

b the **b** option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch, in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

c The **c** option controls key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, keyclick will be disabled. If a value from 0 to 100 is given, it is used to indicate volume, as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp= *path,...*

The **fp=** sets the font path to the directories given in the path argument. The directories are interpreted by the server, not by the client, and are server-dependent. Directories that do not contain font databases created by *mkfontdir* will be ignored by the server.

fp *default*

The *default* argument causes the font path to be reset to the server's default.

fp *rehash*
: The *rehash* argument causes the server to reread the font databases in the current font path. This is generally only used when adding new fonts to a font directory (after running *mkfontdir* to recreate the font database).

–fp or **fp–**
: The **–fp** and **fp–** options remove elements from the current font path. They must be followed by a comma-separated list of directories.

+fp or **fp+**
: This **+fp** and **fp+** options prepend and append elements to the current font path, respectively. They must be followed by a comma-separated list of directories.

led
: The **led** option controls the keyboard LEDs. This controls the turning on or off of one or all of the LEDs. It accepts an optional integer, a preceding dash(-) or an 'on/off' flag. If no parameter or the 'on' flag is given, all LEDs are turned on. If a preceding dash or the flag 'off' is given, all LEDs are turned off. If a value between 1 and 32 is given, that LED will be turned on or off depending on the existence of a preceding dash. A common LED which can be controlled is the "Caps Lock" LED. "xset led 3" would turn led #3 on. "xset -led 3" would turn it off. The particular LED values may refer to different LEDs on different hardware.

m
: The **m** option controls the mouse parameters. The parameters for the mouse are 'acceleration' and 'threshold'. The mouse, or whatever pointer the machine is connected to, will go 'acceleration' times as fast when it travels more than 'threshold' pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen in a flick of the wrist when desired. One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.

p
: The **p** option controls pixel color values. The parameters are the color map entry number in decimal, and a color specification. The root background colors may be changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and 1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.

r — The r option controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.

s — The s option lets you set the screen saver parameters. This option accepts up to two numerical parameters, a 'blank/noblank' flag, an 'expose/noexpose' flag, an 'on/off' flag, or the 'default' flag. If no parameters or the 'default' flag is used, the system will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.

q — The q option gives you information on the current settings.

These settings will be reset to default values when you log out.

Note that not all X implementations are guaranteed to honor all of these options.

SEE ALSO

X(1), Xserver(1), xmodmap(1), xrdb(1), xsetroot(1)

COPYRIGHT

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science
David Krikorian, MIT Project Athena (X11 version)

NAME

xsetroot – root window parameter setting utility for X

SYNOPSIS

xsetroot [-help] [-def] [-display *display*] [-cursor *cursorfile maskfile*] [-bitmap *filename*] [-mod *x y*] [-gray] [-grey] [-fg *color*] [-bg *color*] [-rv] [-solid *color*] [-name *string*]

DESCRIPTION

The *setroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if *-def* is specified, the window is reset to its default state. The *-def* option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (-solid, -gray, -grey, -bitmap, and -mod) may be specified at a time.

OPTIONS

The various options are as follows:

-help Print a usage message and exit.

-def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*
This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be made with the *bitmap(1)* program. You probably want the mask file to be all black until you get used to the way masks work.

-bitmap *filename*
Use the bitmap specified in the file to set the window pattern. You can make your own bitmap files (little pictures) using the *bitmap(1)* program. The entire background will be made up of repeated "tiles" of the bitmap.

-mod *x y*
This is used if you want a plaid-like grid pattern on your screen. x and y are integers ranging from 1 to 16. Try the different combinations. Zero and negative numbers are taken as 1.

-gray Make the entire background gray. (Easier on the eyes.)

-grey Make the entire background grey.

-fg *color*
Use "color" as the foreground color. Foreground and background colors are meaningful only in combination with -cursor, -bitmap, or -mod.

-bg *color*
Use "color" as the background color.

-rv This exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.

-solid *color*
This sets the background of the root window to the specified color. This option is only useful on color servers.

-name *string*
Set the name of the root window to "string". There is no default value. Usually a name is assigned to a window so that the window manager can use a text representation when the window is iconified. This option is unused since you can't iconify the background.

-display *display*
Specifies the server to connect to; see *X(1)*.

SEE ALSO

X(1), xset(1), xrdb(1)

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

Xsgi - SGI Iris server for the X Window System

SYNOPSIS

Xsgi [:displaynumber] [-option ...]

DESCRIPTION

Xsgi is the name for the Silicon Graphics, Inc. X Window System server. The X Windows server provides an interface for programs written for the X Window System, Version 11, developed jointly by MIT's Project Athena and Digital Equipment Corporation, with contributions from many other companies. It was designed by Robert Scheifler, Jim Gettys, and others at MIT, and was based in part on the "W" windowing package developed by Paul Asente at Stanford University.

It is strongly recommended that you refer to the *X Window System User's Guide for Version 11*, by Tim O'Reilly, Valerie Quercia, and Linda Lamb (O'Reilly & Associates, ISBN 0-937175-29-3). Although this man page gives you all the information required to run X clients, the *X Window System User's Guide* contains additional information that is particularly useful to people who intend to develop their own clients.

In addition, you may find the following books to be useful: *Xlib Programming Manual (Volume I)*, by Adrian Nye, O'Reilly & Associates, ISBN 0-937175-26-9 *Xlib Reference Manual (Volume II)*, O'Reilly & Associates, ISBN 0-937175-27-7

STARTING THE SERVER

The X11 Window System runs on the same screen as the NeWS^{TM} window system, which allows GL, NeWS, and X11 clients to run simultaneously. However, there are some differences.

Xsgi is normally run by the *xstart* program during the NeWS login process. However, it may also be started by hand. For more information on starting X from NeWS, please refer to the *xstart(1)* man page.

OPTIONS

Xsgi accepts the following command line options:

−a *number-of-pixels*
Set the mouse acceleration threshold. Currently not supported.

−t *number-of-pixels*
Set the mouse threshold. Currently not supported.

−auth string
Select authorization file.

bc Enable backwards compatibility mode. Currently not supported.

–bs disables backing store support on all screens.

–c Turns off key-click. Currently not supported.

c *percent*
Key-click volume (0-100). Currently not supported.

–r Turns off auto-repeat. **r** Turns on auto-repeat.

–cc *visual-number*
Set the default color visual class.

–f *percent*
Set the bell base volume.

–logo turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

nologo turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

–p *minutes*
sets screen-saver pattern cycle time in minutes. The default is 10 minutes.

–s *minutes*
sets screen-saver timeout time in minutes. The screen saver may be disabled by setting the timeout to 0. This is the default.

–to *seconds*
sets default screensaver timeout in seconds. The screen saver may be disabled by setting the timeout to 0. This is the default.

v Turn on video blanking for screen-saver. Currently not supported.

–v Turn off video blanking for screen-saver. Currently not supported.

–su disables save under support on all screens.

–co *filename*
sets name of RGB color database. The default is */usr/lib/X11/rbg.txt*

–help prints a usage message

–fp *fontPath*
sets the search path for fonts. This defaults to */usr/lib/X11/fonts/misc/*, */usr/lib/X11/fonts/100dpi/*, */usr/lib/X11/fonts/75dpi/*, */usr/lib/fmfonts/*

–fc *cursorFont*
: sets default cursor font. This defaults to *cursor.*

–fn *font* sets the default font. This defaults to *fixed.*

–x *extension-name*
: Loads the named extension at init time.

–wm forces the default backing-store of all windows to be WhenMapped; a not very good way of getting backing-store to apply to all windows.

–static Set the default visual to an 8-bit StaticColor visual. All 256 entries in the default colormap are preallocated.

–gl Sets the default visual to an 8-bit PseudoColor visual with the colors 0-15 and 32-63 in the default colormap preallocated for use by the gl. This leaves 208 writable color cells for X.

–envm Sets the default visual to an 8-bit PseudoColor visual with all colors in the default colormap except 16-31 preallocated. GL and NeWS clients consider these colors off limits.

–pseudo The default visual is set to an 8-bit PseudoColor visual with no colors in the default colormap preallocated. There is no color map arbitration provided. GL and NeWS clients may also overwrite the same colors.

SECURITY

Xsgi uses an access control list for deciding whether or not to accept connections from clients on a particular machine. This list initially consists of the host on which the server is running as well as any machines listed in the file */etc/X***n***.hosts*, where **n** is the display number of the server. Each line of the file should contain an Internet hostname (e.g. expo.lcs.mit.edu).

Users can add or remove hosts from this list and enable or disable access control using the *xhost* command from the same machine as the server. Please refer to the *xhost(1)* man page for more information.

Unlike some window systems, X does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen. See also *xauth(1)*.

SIGNALS

The sample server attaches special meaning to the following signals:

SIGHUP This signal causes *Xsgi* to close all existing connections, free all resources, and restore all defaults. It is sent by the display manager whenever the main user's main application (usually an *xterm* or window manager) exits to force the server to clean up and prepare for the next user.

SIGTERM
This signal causes the *Xsgi* to exit cleanly.

SEE ALSO

X(1), Xserver(1), xdm(1), mkfontdir(1), xinit(1), xterm(1), twm(1), xhost(1), xset(1), xsetroot(1), xstart(1), xauth(1)

BUGS

The option syntax is inconsistent with itself and *xset(1)*.

If *X* dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The -a, bc, -c, c, -t, ttyxx, v, and -v options are not supported.

Backing store does not work correctly unless an *X* window manager is running. If an *X* window manager (eg. *twm, mwm*) is not used, then the -bs option should be used.

Xsgi ignores the CAPSLOCK key.

ADDITIONAL DOCUMENTATION

Additional documentation for developers of X Window System clients is available on-line and directly from MIT. To obtain materials directly from MIT, contact:

MIT Software Distribution Center
Technology Licensing Office
MIT E32-300
77 Massachusetts Avenue
Cambridge, MA 02139

Their telephone number is (617) 253-6966, and the "X Ordering Hotline" is (617) 258-8330.

COPYRIGHT

Copyright 1989, 1990, Silicon Graphics Inc.
Copyright 1987, 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

NAME

xshowcmap – show colormap

SYNOPSIS

xshowcmap [options]

DESCRIPTION

Xshowcmap displays the contents of the currently active colormap in a window. The created window shows a square for every color currently defined in the servers active colormap. The number of squares is the number of colormap-cells the server supports.

Xshowcmap has been specially written to aid server debugging/verification.

To leave *xshowcmap* type 'q' while the cursor is in its window.

The following options are valid:

-bd color
as usual - change border color

-bw number
as usual - change border width

host:display
Name of the display.

BUGS

There might be some - it has been tested with Siemens servers running both monochrome and color, and with uVax Color-Workstations running color.

AUTHORS

Claus Gittinger (..!decvax!unido!sinix!claus)

NAME

X Standards

SYNOPSIS

The major goal of the MIT X Consortium is to promote cooperation within the computer industry in the creation of standard software interfaces at all layers in the X Window System environment. The status of various standards and proposed standards, and of the software in the X11R4 distribution, is explained below.

STANDARDS

The following documents are MIT X Consortium standards:

X Window System Protocol
X Version 11, Release 4
Robert W. Scheifler

Xlib - C Language X Interface
X Version 11, Release 4
James Gettys, Robert W. Scheifler, Ron Newman

X Toolkit Intrinsics - C Language Interface
X Version 11, Release 4
Joel McCormack, Paul Asente, Ralph R. Swick

Bitmap Distribution Format
Version 2.1

Inter-Client Communication Conventions Manual
Version 1.0
David S. H. Rosenthal

Compound Text Encoding
Version 1.1
Robert W. Scheifler

X Logical Font Description Conventions
Version 1.3
Jim Flowers

X Display Manager Control Protocol
Version 1.0
Keith Packard

X11 Nonrectangular Window Shape Extension

Version 1.0
Keith Packard

DRAFT STANDARDS

The following documents are draft standards of the MIT X Consortium. To become standards, further ''proof of concept'' is required, in the form of working implementations. The specifications may be subject to incompatible changes if implementation efforts uncover significant problems.

PEX Protocol Specification
Version 4.0P
Randi J. Rost (editor)

Extending X for Double-Buffering, Multi-Buffering, and Stereo
Version 3.2
Jeffrey Friedberg, Larry Seiler, Jeff Vroom

PUBLIC REVIEW DRAFTS

The following documents are out for Public Review for adoption as MIT X Consortium standards.

X11 Input Extension Protocol Specification
Public Review Draft
George Sachs, Mark Patrick

X11 Input Extension Library Specification
Public Review Draft
Mark Patrick, George Sachs

INCLUDE FILES

The following include files are part of the Xlib standard. The C++ support in these header files is experimental, and is not yet part of the standard.

<X11/X.h>
<X11/Xatom.h>
<X11/Xproto.h>
<X11/Xprotostr.h>
<X11/keysym.h>
<X11/keysymdef.h>
<X11/Xlib.h>
<X11/Xresource.h>
<X11/Xutil.h>
<X11/cursorfont.h>

<X11/X10.h>
<X11/Xlibint.h>

The following include files are part of the X Toolkit Intrinsics standard. The C++ support in these header files is experimental, and is not yet part of the standard.

<X11/Composite.h>
<X11/CompositeP.h>
<X11/ConstrainP.h>
<X11/Constraint.h>
<X11/Core.h>
<X11/CoreP.h>
<X11/Intrinsic.h>
<X11/IntrinsicP.h>
<X11/Object.h>
<X11/ObjectP.h>
<X11/Quarks.h>
<X11/RectObj.h>
<X11/RectObjP.h>
<X11/Shell.h>
<X11/ShellP.h>
<X11/StringDefs.h>
<X11/Vendor.h>
<X11/VendorP.h>

The following include file is part of the Nonrectangular Window Shape Extension standard.

<X11/extensions/shape.h>

The following include file is part of the Multi-Buffering draft standard.

<X11/extensions/multibuf.h>

NON STANDARDS

The X11R4 distribution contains *sample* implementations, not *reference* implementations. Although much of the code is believed to be correct, the code should be assumed to be in error wherever it conflicts with the specification.

At the public release of X11R4, the only MIT X Consortium standards are the ones listed above. No other documents, include files, or software in X11R4 carry special status within the X Consortium. For example, none of the following are standards: internal interfaces of the sample server; the MIT-SHM extension; the Input Synthesis extension; the Athena Widget Set;

the Xmu library; the Xau library; CLX, the Common Lisp interface to X (although a Consortium review is finally expected to begin); the RGB database; the fonts distributed with X11R4; the applications distributed with X11R4; the include files <X11/XWDFile.h> and <X11/Xos.h>; the bitmap files in <X11/bitmaps>.

NAME

xstart – start up the sgi X server as a NeWS client

SYNOPSIS

xstart [**-c**] [*X command*]

DESCRIPTION

xstart will start the sgi X server in background. It is intended to be executed from *init.ps,* the PostScript file which gets executed during the bringup of *SGI's NeWS*-based window system. However, *xstart* usage is not confined to init.ps; it can be executed at any time so long as *NeWS* is running.

If *xstart* is given the –c option, it will conditionally start up X, depending on whether *xSGINeWS* is configured. The command

chkconfig xSGINeWS on

will configure *xSGINeWS* on a per-machine basis. However, if the user has a file *$HOME/.xSGINeWS* containing either "on" or "off," that file will override the system-wide config option.

The actual command which *xstart* will use for starting X is determined by (in order of decreasing priority):

1. If there are arguments to *xstart* other than -c, those arguments specify the command.

2. If there are no arguments to *xstart* (except for possibly -c), the command is taken from the ascii file *$HOME/.xSGINeWS.cmd.*

3. If *$HOME/.xSGINeWS.cmd* doesn't exist, the command is taken from the ascii file */etc/gl/xSGINeWS.cmd.*

4. If */etc/gl/.xSGINeWS.cmd* doesn't exist, the default command used by *xstart* is

/usr/bin/X11/Xsgi –bs –envm.

Before starting X, *xstart* changes the current working directory to /tmp. Hence care should be taken if the user supplies an X server command (via *xstart* arguments or one of the *.cmd* files), and if the X server name does not include a full pathname.

A side effect of starting X by *xstart* is that X server will log its error messages to */usr/adm/X0msgs*, provided that file is writable.

WARNING

The purpose of *xstart* is simply to conditionally start up *Xsgi* from *init.ps.* At some point in the future, *xstart* will not be applicable and will no longer be supported.

NAME

xstdcmap - X standard colormap utility

SYNOPSIS

xstdcmap [-all] [-best] [-blue] [-default] [-delete *map*] [-display *display*] [-gray] [-green] [-help] [-red] [-verbose]

DESCRIPTION

The *xstdcmap* utility can be used to selectively define standard colormap properties. It is intended to be run from a user's X startup script to create standard colormap definitions in order to facilitate sharing of scarce colormap resources among clients. Where at all possible, colormaps are created with read-only allocations.

OPTIONS

The following options may be used with *xstdcmap*:

—all This option indicates that all six standard colormap properties should be defined on each screen of the display. Not all screens will support visuals under which all six standard colormap properties are meaningful. *xstdcmap* will determine the best allocations and visuals for the colormap properties of a screen. Any previously existing standard colormap properties will be replaced.

—best This option indicates that the RGB_BEST_MAP should be defined.

—blue This option indicates that the RGB_BLUE_MAP should be defined.

—default This option indicates that the RGB_DEFAULT_MAP should be defined.

—delete *map*
This option specifies that a standard colormap property should be removed. *mapP may be one of: default, best, red, green, blue, or gray.*

—display *display*
This option specifies the host and display to use; see *X(1).*

—gray This option indicates that the RGB_GRAY_MAP should be defined.

—green This option indicates that the RGB_GREEN_MAP should be defined.

—help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xstdcmap*.

−red This option indicates that the RGB_RED_MAP should be defined.

−verbose
This option indicates that *xstdcmap* should print logging information as it parses its input and defines the standard colormap properties.

ENVIRONMENT

DISPLAY
to get default host and display number.

SEE ALSO

X(1)

COPYRIGHT

AUTHOR

Donna Converse, MIT X Consortium

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**–v**] [[**–c**] [**–**] [file]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only. The –v flag makes *xstr* verbose.

The command

```
xstr –c name
```

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

```
xstr
```

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

```
xstr name
```

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '–' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc –E name.c | xstr –c –
cc –c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

SEE ALSO

mkstr(1)

AUTHOR

William Joy

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

xterm – terminal emulator for X

SYNOPSIS

xterm [-*toolkitoption* ...] [-option ...]

DESCRIPTION

The *xterm* program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), *xterm* will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the ''active'' window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be chosen through escape sequences, the ''Modes'' menu in the VT102 window, and the ''Tektronix'' menu in the 4014 window.

OPTIONS

The *xterm* terminal emulator accepts all of the standard X Toolkit command line options as well as the following (if the option begins with a '+' instead of a '–', the option is restored to its default value):

-help This causes *xterm* to print out a verbose message describing its options.

-132 Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the *xterm* window will resize appropriately.

-ah This option indicates that *xterm* should always highlight the text cursor and borders. By default, *xterm* will display a hollow text cursor whenever the focus is lost or the pointer leaves the window.

+ah This option indicates that *xterm* should do text cursor highlighting.

−b *number*
: This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.

−cc *characterclassrange*:*value*[,...]
: This sets classes indicated by the given ranges for using in selecting by words. See the section specifying character classes.

−cn This option indicates that newlines should not be cut in line-mode selections.

+cn This option indicates that newlines should be cut in line-mode selections.

−cr *color*
: This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

−cu This option indicates that *xterm* should work around a bug in the *curses*(3x) cursor motion package that causes the *more*(1) program to display lines that are exactly the width of the window and are followed by a line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu This option indicates that that *xterm* should not work around the *curses*(3x) bug mentioned above.

−e *program [arguments ...]*
: This option specifies the program (and its command line arguments) to be run in the *xterm* window. It also sets the window title and icon name to be the basename of the program being executed if neither *-T* nor *-n* are given on the command line. **This must be the last option on the command line.**

−fb *font* This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default is to do overstriking of the normal font.

−j This option indicates that *xterm* should do jump scrolling. Normally, text is scrolled one line at a time; this option allows *xterm* to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.

+j This option indicates that *xterm* should not do jump scrolling.

–l This option indicates that *xterm* should send all terminal output to a log file as well as to the screen. This option can be enabled or disabled using the "xterm X11" menu.

+l This option indicates that *xterm* should not do logging.

–lf *filename*
This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (|), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*) and is created in the directory from which *xterm* was started (or the user's home directory in the case of a login window).

–ls This option indicates that the shell that is started in the *xterm* window be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be a normal "subshell").

–mb This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "Modes" menu.

+mb This option indicates that margin bell should not be rung.

–mc milliseconds
This option specifies the maximum time between multi-click selections.

–ms *color*
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.

–nb *number*
This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.

–rw This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines and is encouraged. This option can be turned on and off from the "Modes" menu.

+rw This option indicates that reverse-wraparound should not be allowed.

−s This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.

+s This option indicates that *xterm* should scroll synchronously.

−sb This option indicates that some number of lines that are scrolled off the top of the window should be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the ``Modes'' menu.

+sb This option indicates that a scrollbar should not be displayed.

−sf This option indicates that Sun Function Key escape codes should be generated for function keys.

+sf This option indicates that the standard escape codes should be generated for function keys.

−si This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the ``Modes'' menu.

+si This option indicates that output to a window should cause it to scroll to the bottom.

−sk This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal position at the bottom of the scroll region.

+sk This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.

−sl *number*
This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.

−t This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the ``Modes'' menus.

+t This option indicates that *xterm* should start in VT102 mode.

—tm *string*
This option specifies a series of terminal setting keywords followed by the characters that should be bound to those functions, similar to the *stty* program. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete.

—tn *name*
This option specifies the name of the terminal type to be set in the TERM environment variable. This terminal type must exist in the *termcap(5)* database and should have *li#* and *co#* entries.

—ut This option indicates that *xterm* shouldn't write a record into the the system log file */etc/utmp.*

+ut This option indicates that *xterm* should write a record into the system log file */etc/utmp.*

—vb This option indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.

+vb This option indicates that a visual bell should not be used.

—wf This option indicates that *xterm* should wait for the window to be mapped the first time before starting the subprocess so that the initial terminal size settings and environment variables are correct. It the application's responsibility to catch subsequent terminal size changes.

+wf This option indicates that *xterm* show not wait before starting the subprocess.

—C This option indicates that this window should receive console output. This is not supported on all systems.

—S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode, plus the number of the inherited file descriptor. The option is parsed "%c%c%d". This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

%geom This option specifies the preferred size and position of the Tektronix window. It is shorthand for specifying the "**tekGeometry*" resource.

#geom This option specifies the preferred position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.

–T *string*
This option specifies the title for *xterm*'s windows. It is equivalent to **-title**.

–n *string*
This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource. Note that this is not the same as the toolkit option **-name** (see below). The default icon name is the application name.

–r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.

–w *number*
This option specifies the width in pixels of the border surrounding the window. It is equivalent to **-borderwidth** or **-bw**.

The following standard X Toolkit command line arguments are commonly used with *xterm*:

–bg *color*
This option specifies the color to use for the background of the window. The default is ''white.''

–bd *color*
This option specifies the color to use for the border of the window. The default is ''black.''

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color*
This option specifies the color to use for displaying text. The default is ''black''.

–fn *font* This option specifies the font to be used for displaying normal text. The default is *fixed*.

–name *name*
This option specifies the application name under which resources are to be obtained, rather than the default executable file name. *Name* should not contain ''.'' or ''*'' characters.

—title *string*
This option specifies the window title string, which may be displayed by window managers if the user so chooses. The default title is the command line specified after the **-e** option, if any, otherwise the application name.

—rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

—geometry *geometry*
This option specifies the preferred size and position of the VT102 window; see *X(1)*.

—display *display*
This option specifies the X server to contact; see *X(1)*.

—xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

—iconic This option indicates that *xterm* should ask the window manager to start it as an icon rather than as the normal window.

RESOURCES

The program understands all of the core X Toolkit resource names and classes as well as:

iconGeometry (class IconGeometry)
Specifies the preferred size and position of the application when iconified. It is not necessarily obeyed by all window managers.

termName (class TermName)
Specifies the terminal type name to be set in the TERM environment variable.

title (class Title)
Specifies a string that may be used by the window manager when displaying this application.

ttyModes (class TtyModes)
Specifies a string containing terminal setting keywords and the characters to which they may be bound. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete. This is very useful for overriding the default terminal settings without having to do an *stty* every time an *xterm* is started.

utmpInhibit (class **UtmpInhibit**)
: Specifies whether or not *xterm* should try to record the user's terminal in */etc/utmp*.

sunFunctionKeys (class **SunFunctionKeys**)
: Specifies whether or not Sun Function Key escape codes should be generated for function keys instead of standard escape sequences.

The following resources are specified as part of the *vt100* widget (class *VT100*):

allowSendEvents (class **AllowSendEvents**)
: Specifies whether or not synthetic key and button events (generated using the X protocol SendEvent request) should be interpreted or discarded. The default is "false" meaning they are discarded. Note that allowing such events creates a very large security hole.

alwaysHighlight (class **AlwaysHighlight**)
: Specifies whether or not *xterm* should always display a highlighted text cursor. By default, a hollow text cursor is displayed whenever the pointer moves out of the window or the window loses the input focus.

boldFont (class **Font**)
: Specifies the name of the bold font to use instead of overstriking.

c132 (class **C132**)
: Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

charClass (class **CharClass**)
: Specifies comma-separated lists of character class bindings of the form [*low*-]*high*:*value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

curses (class **Curses**)
: Specifies whether or not the last column bug in *curses*(3x) should be worked around. The default is "false."

background (class **Background**)
: Specifies the color to use for the background of the window. The default is "white."

foreground (class Foreground)
: Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the "text" color change color. The default is "black."

cursorColor (class Foreground)
: Specifies the color to use for the text cursor. The default is "black."

eightBitInput (class EightBitInput)
: Specifies whether or not eight-bit characters should be accepted. The default is "true."

font (class Font)
: Specifies the name of the normal font. The default is "vtsingle."

font1 (class Font1)
: Specifies the name of the first alternate font.

font2 (class Font2)
: Specifies the name of the second alternate font.

font3 (class Font3)
: Specifies the name of the third alternate font.

font4 (class Font4)
: Specifies the name of the fourth alternate font.

geometry (class Geometry)
: Specifies the preferred size and position of the VT102 window.

internalBorder (class BorderWidth)
: Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class JumpScroll)
: Specifies whether or not jump scroll should be used. The default is "true".

logFile (class Logfile)
: Specifies the name of the file to which a terminal session is logged. The default is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*).

logging (class Logging)
: Specifies whether or not a terminal session should be logged. The default is "false."

logInhibit (class **LogInhibit**)
: Specifies whether or not terminal session logging should be inhibited. The default is "false."

loginShell (class **LoginShell**)
: Specifies whether or not the shell to be run in the window should be started as a login shell. The default is "false."

marginBell (class **MarginBell**)
: Specifies whether or not the bell should be run when the user types near the right margin. The default is "false."

multiScroll (class **MultiScroll**)
: Specifies whether or not asynchronous scrolling is allowed. The default is "false."

multiClickTime (class **MultiClickTime**)
: Specifies the maximum time in milliseconds between multi-clock select events. The default is 250 milliseconds.

multiScroll (class **MultiScroll**)
: Specifies whether or not scrolling should be done asynchronously. The default is "false."

nMarginBell (class **Column**)
: Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.

pointerColor (class **Foreground**)
: Specifies the foreground color of the pointer. The default is "XtDefaultForeground."

pointerColorBackground (class **Background**)
: Specifies the background color of the pointer. The default is "XtDefaultBackground."

pointerShape (class **Cursor**)
: Specifies the name of the shape of the pointer. The default is "xterm."

reverseVideo (class **ReverseVideo**)
: Specifies whether or not reverse video should be simulated. The default is "false."

reverseWrap (class **ReverseWrap**)
: Specifies whether or not reverse-wraparound should be enabled. The default is "false."

saveLines (class SaveLines)
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class ScrollBar)
Specifies whether or not the scrollbar should be displayed. The default is "false."

scrollInput (class ScrollCond)
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class ScrollCond)
Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

scrollLines (class ScrollLines)
Specifies the number of lines that the *scroll-back* and *scroll-forw* actions should use as a default. The default value is 1.

signalInhibit (class SignalInhibit)
Specifies whether or not the entries in the "xterm X11" menu for sending signals to *xterm* should be disallowed. The default is "false."

tekGeometry (class Geometry)
Specifies the preferred size and position of the Tektronix window.

tekInhibit (class TekInhibit)
Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekSmall (class TekSmall)
Specifies whether or not the Tektronix mode window should start in its smallest size if no explicit geometry is given. This is useful when running *xterm* on displays with small screens. The default is "false."

tekStartup (class TekStartup)
Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class TiteInhibit)
Specifies whether or not *xterm* should remove remove *ti* or *te* termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string.

translations (class Translations)
: Specifies the key and button bindings for menus, selections, ``programmed strings'', etc. See **ACTIONS** below.

visualBell (class VisualBell)
: Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is ``false.''

waitForMap (class WaitForMap)
: Specifies whether or not *xterm* should wait for the initial window map before starting the subprocess. The default is ``false.''

The following resources are specified as part of the *tek4014* widget (class *Tek4014*):

width (class Width)
: Specifies the width of the Tektronix window in pixels.

height (class Height)
: Specifies the height of the Tektronix window in pixels.

fontLarge (class Font)
: Specifies the large font to use in the Tektronix window.

font2 (class Font)
: Specifies font number 2 to use in the Tektronix window.

font3 (class Font)
: Specifies font number 2 font to use in the Tektronix window.

fontSmall (class Font)
: Specifies the small font to use in the Tektronix window.

The resources that may be specified for the various menus are described in the documentation for the Athena **SimpleMenu** widget. The name and classes of the entries in each of the menus are listed below.

The *mainMenu* has the following entries:

securekbd (class SmeBSB)
: This entry invokes the **secure()** action.

allowsends (class SmeBSB)
: This entry invokes the **allow-send-events(toggle)** action.

logging (class SmeBSB)
: This entry invokes the **set-logging(toggle)** action.

redraw (class SmeBSB)
: This entry invokes the **redraw()** action.

line1 (class SmeLine)
: This is a separator.

suspend (class SmeBSB)
: This entry invokes the **send-signal(suspend)** action on systems that support job control.

continue (class SmeBSB)
: This entry invokes the **send-signal(cont)** action on systems that support job control.

interrupt (class SmeBSB)
: This entry invokes the **send-signal(int)** action.

hangup (class SmeBSB)
: This entry invokes the **send-signal(hup)** action.

terminate (class SmeBSB)
: This entry invokes the **send-signal(term)** action.

kill (class SmeBSB)
: This entry invokes the **send-signal(kill)** action.

line2 (class SmeLine)
: This is a separator.

quit (class SmeBSB)
: This entry invokes the **quit()** action.

The *vtMenu* has the following entries:

scrollbar (class SmeBSB)
: This entry invokes the **set-scrollbar(toggle)** action.

jumpscroll (class SmeBSB)
: This entry invokes the **set-jumpscroll(toggle)** action.

reversevideo (class SmeBSB)
: This entry invokes the **set-reverse-video(toggle)** action.

autowrap (class SmeBSB)
: This entry invokes the **set-autowrap(toggle)** action.

reversewrap (class SmeBSB)
: This entry invokes the **set-reversewrap(toggle)** action.

autolinefeed (class **SmeBSB**)
: This entry invokes the **set-autolinefeed(toggle)** action.

appcursor (class **SmeBSB**)
: This entry invokes the **set-appcursor(toggle)** action.

appkeypad (class **SmeBSB**)
: This entry invokes the **set-appkeypad(toggle)** action.

scrollkey (class **SmeBSB**)
: This entry invokes the **set-scroll-on-key(toggle)** action.

scrollttyoutput (class **SmeBSB**)
: This entry invokes the **set-scroll-on-tty-output(toggle)** action.

allow132 (class **SmeBSB**)
: This entry invokes the **set-allow132(toggle)** action.

cursesemul (class **SmeBSB**)
: This entry invokes the **set-cursesemul(toggle)** action.

visualbell (class **SmeBSB**)
: This entry invokes the **set-visualbell(toggle)** action.

marginbell (class **SmeBSB**)
: This entry invokes the **set-marginbell(toggle)** action.

altscreen (class **SmeBSB**)
: This entry is currently disabled.

line1 (class **SmeLine**)
: This is a separator.

softreset (class **SmeBSB**)
: This entry invokes the **soft-reset()** action.

hardreset (class **SmeBSB**)
: This entry invokes the **hard-reset()** action.

line2 (class **SmeLine**)
: This is a separator.

tekshow (class **SmeBSB**)
: This entry invokes the **set-visibility(tek,toggle)** action.

tekmode (class **SmeBSB**)
: This entry invokes the **set-terminal-type(tek)** action.

vthide (class **SmeBSB**)
: This entry invokes the **set-visibility(vt,off)** action.

The *fontMenu* has the following entries:

fontdefault (class SmeBSB)
: This entry invokes the **set-vt-font(d)** action.

font1 (class SmeBSB)
: This entry invokes the **set-vt-font(1)** action.

font2 (class SmeBSB)
: This entry invokes the **set-vt-font(2)** action.

font3 (class SmeBSB)
: This entry invokes the **set-vt-font(3)** action.

font4 (class SmeBSB)
: This entry invokes the **set-vt-font(4)** action.

fontescape (class SmeBSB)
: This entry invokes the **set-vt-font(e)** action.

fontsel (class SmeBSB)
: This entry invokes the **set-vt-font(s)** action.

The *tekMenu* has the following entries:

tektextlarge (class SmeBSB)
: This entry invokes the **set-tek-text(l)** action.

tektext2 (class SmeBSB)
: This entry invokes the **set-tek-text(2)** action.

tektext3 (class SmeBSB)
: This entry invokes the **set-tek-text(3)** action.

tektextsmall (class SmeBSB)
: This entry invokes the **set-tek-text(s)** action.

line1 (class SmeLine)
: This is a separator.

tekpage (class SmeBSB)
: This entry invokes the **tek-page()** action.

tekreset (class SmeBSB)
: This entry invokes the **tek-reset()** action.

tekcopy (class SmeBSB)
: This entry invokes the **tek-copy()** action.

line2 (class SmeLine)
: This is a separator.

vtshow (class **SmeBSB**)
: This entry invokes the **set-visibility(vt,toggle)** action.

vtmode (class **SmeBSB**)
: This entry invokes the **set-terminal-type(vt)** action.

tekhide (class **SmeBSB**)
: This entry invokes the **set-visibility(tek,toggle)** action.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness**)
: Specifies the width in pixels of the scrollbar.

background (class **Background**)
: Specifies the color to use for the background of the scrollbar.

foreground (class **Foreground**)
: Specifies the color to use for the foreground of the scrollbar. The ''thumb'' of the scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include ''xterm'', ''vt102'', ''vt100'' and ''ansi'', and *xterm* automatically searches the termcap file in this order for these entries and then sets the ''TERM'' and the ''TERMCAP'' environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the *''Xterm Control Sequences''* document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the **Tektronix** menu; see below). The name of the file will be ''COPY*yy–MM–dd.hh:mm:ss*'', where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key. The assignment of the functions described below to keys and buttons may be changed through the resource database; see ACTIONS below.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer and made the PRIMARY selection when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection. If the key/button bindings specify that an X selection is to be made, *xterm* will leave the selected text highlighted for as long as it is the selection owner.

Pointer button two (usually middle) 'types' (pastes) the text from the PRIMARY selection, if any, otherwise from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, that is you can swap "right" and "left" everywhere in the rest of this paragraph...) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window dows not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer button is pressed, the corresponding upper case letter is sent. To distinguish a pointer button from a key, the high bit of the character is set (but this is bit is normally stripped unless the terminal mode is RAW; see *tty*(4) for details).

MENUS

Xterm has four menus, named *mainMenu*, *vtMenu*, *fontMenu*, and *tekMenu*. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two section, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The *mainMenu* contains items that apply to both the VT102 and Tektronix windows. The **Secure Keyboard** mode is be used when typing in passwords or other sensitive data in an unsecure environment; see **SECURITY** below. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The *vtMenu* sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to their initial states just after *xterm* has finished processing the command line options.

The *fontMenu* sets the font used in the VT102 window.

The *tekMenu* sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

SECURITY

X environments differ in their security consciousness. MIT servers, run under *xdm*, are capable of using a "magic cookie" authorization scheme that can provide a reasonable level of security for many people. If your server is only using a host-based mechanism to control access to the server (see *xhost(1)*), then if you enable access for a host and other users are also permitted to run clients on that same host, there is every possibility that someone can run an application that will use the basic services of the X protocol to snoop on your activities, potentially capturing a transcript of everything you type at the keyboard. This is of particular concern when you want to type in a password or other sensitive data. The best solution to this problem is to use a better authorization mechanism that host-based control, but a simple mechanism exists for protecting keyboard input in *xterm*.

The **xterm** menu (see **MENUS** above) contains a **Secure Keyboard** entry which, when enabled, ensures that all keyboard input is directed *only* to *xterm* (using the GrabKeyboard protocol request). When an application prompts you for a password (or other sensitive data), you can enable **Secure Keyboard** using the menu, type in the data, and then disable **Secure Keyboard** using the menu again. Only one X client at a time can secure the keyboard, so when you attempt to enable **Secure Keyboard** it may fail. In this case, the bell will sound. If the **Secure Keyboard** succeeds, the foreground and background colors will be exchanged (as if you selected the **Reverse Video** entry in the **Modes** menu); they will be exchanged again when you exit secure mode. If the colors do *not* switch, then you should be *very* suspicious that you are being spoofed. If the application you are running displays a prompt before asking for the password, it is safest to enter secure mode *before* the prompt gets displayed, and to make sure that the prompt gets displayed correctly (in the new colors), to minimize the probability of spoofing. You can also bring up the menu again and make sure that a check mark appears next to the entry.

Secure Keyboard mode will be disabled automatically if your xterm window becomes iconified (or otherwise unmapped), or if you start up a reparenting window manager (that places a title bar or other decoration around the window) while in **Secure Keyboard** mode. (This is a feature of the X protocol not easily overcome.) When this happens, the foreground and background colors will be switched back and the bell will sound in warning.

CHARACTER CLASSES

Clicking the middle mouse button twice in rapid succession will cause all characters of the same class (e.g. letters, white space, punctuation) to be selected. Since different people have different preferences for what should be selected (for example, should filenames be selected as a whole or only the separate subnames), the default mapping can be overridden through the use of the *charClass* (class *CharClass*) resource.

This resource is simply a list of *range*:*value* pairs where the range is either a single number or *low-high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. The *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The default table is:

```
static int charClass[128] = {
/* NUL  SOH  STX  ETX  EOT  ENQ  ACK  BEL */
  32,   1,   1,   1,   1,   1,   1,   1,
/* BS   HT   NL   VT   NP   CR   SO   SI */
   1,  32,   1,   1,   1,   1,   1,   1,
/* DLE  DC1  DC2  DC3  DC4  NAK  SYN  ETB */
   1,   1,   1,   1,   1,   1,   1,   1,
/* CAN   EM  SUB  ESC   FS   GS   RS   US */
   1,   1,   1,   1,   1,   1,   1,   1,
/*  SP    !    "    #    $    %    &    ' */
  32,  33,  34,  35,  36,  37,  38,  39,
/*   (    )    *    +    ,    -    .    / */
  40,  41,  42,  43,  44,  45,  46,  47,
/*   0    1    2    3    4    5    6    7 */
  48,  48,  48,  48,  48,  48,  48,  48,
/*   8    9    :    ;    <    =    >    ? */
  48,  48,  58,  59,  60,  61,  62,  63,
/*   @    A    B    C    D    E    F    G */
  64,  48,  48,  48,  48,  48,  48,  48,
/*   H    I    J    K    L    M    N    O */
  48,  48,  48,  48,  48,  48,  48,  48,
/*   P    Q    R    S    T    U    V    W */
  48,  48,  48,  48,  48,  48,  48,  48,
/*   X    Y    Z    [    \    ]    ^    _ */
  48,  48,  48,  91,  92,  93,  94,  48,
/* `   a   b   c   d   e   f   g*/
  96, 48, 48, 48, 48, 48, 48, 48,
/* h   i   j   k   l   m   n   o*/
  48, 48, 48, 48, 48, 48, 48, 48,
/* p   q   r   s   t   u   v   w*/
  48, 48, 48, 48, 48, 48, 48, 48,
/* x   y   z   {   |   }   ~ DEL*/
  48, 48, 48,123,124,125,126,  1};
```

For example, the string "33:48,37:48,45-47:48,64:48" indicates that the exclamation mark, percent sign, dash, period, slash, and ampersand characters should be treated the same way as characters and numbers. This is very useful for cutting and pasting electronic mailing addresses and filenames.

ACTIONS

It is possible to rebind keys (or sequences of keys) to arbitrary strings for input, by changing the translations for the vt100 or tek4014 widgets. Changing the translations for events other than key and button events is not expected, and will cause unpredictable behavior. The following actions are provided for using within the *vt100* or *tek4014* **translations** resources:

bell([*percent*])
: This action rings the keyboard bell at the specified percentage above or below the base volume.

ignore() This action ignores the event but checks for special pointer position escape sequences.

insert() This action is a synonym for **insert-seven-bit**()

insert-seven-bit()
: This action inserts the 7-bit USASCII character or string associated with the keysym that was pressed.

insert-eight-bit()
: This action inserts the 8-bit ISO Latin-1 character or string associated with the keysym that was pressed.

insert-selection(*sourcename* [, ...])
: This action inserts the string found in the selection or cutbuffer indicated by *sourcename*. Sources are checked in the order given (case is significant) until one is found. Commonly-used selections include: *PRIMARY*, *SECONDARY*, and *CLIPBOARD*. Cut buffers are typically named *CUT_BUFFER0* through *CUT_BUFFER7*.

keymap(*name*)
: This action dynamically defines a new translation table whose resource name is *name* with the suffix *Keymap* (case is significant). The name *None* restores the original translation table.

popup-menu(*menuname*)
: This action displays the specified popup menu. Valid names (case is significant) include: *mainMenu*, *vtMenu*, *fontMenu*, and *tekMenu*.

secure() This action toggles the *Secure Keyboard* mode described in the section named **SECURITY**, and is invoked from the **securekbd** entry in *mainMenu*.

select-start()
: This action begins text selection at the current pointer location. See the section on **POINTER USAGE** for information on making selections.

select-extend()
: This action tracks the pointer and extends the selection. It should only be bound to Motion events.

select-end(*destname* [, ...])
: This action puts the currently selected text into all of the selections or cutbuffers specified by *destname*.

select-cursor-start()
: This action is similar to **select-start** except that it begins the selection at the current text cursor position.

select-cursor-end(*destname* [, ...])
: This action is similar to **select-end** except that it should be used with **select-cursor-start**.

set-vt-font(*d/1/2/3/4/e/s* [,*normalfont* [, *boldfont*]])
: This action sets the font or fonts currently being used in the VT102 window. The first argument is a single character that specifies the font to be used: *d* or *D* indicate the default font (the font initially used when *xterm* was started), *1* through *4* indicate the fonts specified by the *font1* through *font4* resources, *e* or *E* indicate the normal and bold fonts that may be set through escape codes (or specified as the second and third action arguments, respectively), and *i* or *I* indicate the font selection (as made by programs such as *xfontsel(1)*) indicated by the second action argument.

start-extend()
This action is similar to **select-start except that the selection is extended to the current pointer location.**

start-cursor-extend()
This action is similar to **select-extend** except that the selection is extended to the current text cursor position.

string(*string***)**
This action inserts the specified text string as if it had been typed. Quotation is necessary if the string contains whitespace or non-alphanumeric characters. If the string argument begins with the characters ''0x'', it is interpreted as a hex character constant.

scroll-back(*count* **[,***units***])**
This action scrolls the text window backward so that text that had previously scrolled off the top of the screen is now visible. The *count* argument indicates the number of *units* (which may be *page*, *halfpage*, *pixel*, or *line*) by which to scroll.

scroll-forw(*count* **[,***units***])**
This action scrolls is similar to **scroll-back** except that it scrolls the other direction.

allow-send-events(*on/off/toggle***)**
This action set or toggles the **allowSendEvents** resource and is also invoked by the **allowsends** entry in *mainMenu.*

set-logging(*on/off/toggle***)**
This action toggles the **logging** resource and is also invoked by the **logging** entry in *mainMenu.*

redraw()
This action redraws the window and is also invoked by the *redraw* entry in *mainMenu.*

send-signal(*signame***)**
This action sends the signal named by *signame* (which may also be a number) to the *xterm* subprocess (the shell or program specified with the *-e* command line option) and is also invoked by the **suspend**, **continue**, **interrupt**, **hangup**, **terminate**, and *kill* entries in *mainMenu*. Allowable signal names are (case is not significant): *suspend*, *tstp* (if supported by the operating system), *cont* (if supported by the operating system), *int*, *hup*, *term*, and *kill.*

quit() This action sends a SIGHUP to the subprogram and exits. It is also invoked by the **quit** entry in *mainMenu.*

set-scrollbar(*on/off/toggle*)
This action toggles the **scrollbar** resource and is also invoked by the **scrollbar** entry in *vtMenu.*

set-jumpscroll(*on/off/toggle*)
This action toggles the **jumpscroll** resource and is also invoked by the **jumpscroll** entry in *vtMenu.*

set-reverse-video(*on/off/toggle*)
This action toggles the *reverseVideo* resource and is also invoked by the **reversevideo** entry in *vtMenu.*

set-autowrap(*on/off/toggle*)
This action toggles automatic wrapping of long lines and is also invoked by the **autowrap** entry in *vtMenu.*

set-reversewrap(*on/off/toggle*)
This action toggles the **reverseWrap** resource and is also invoked by the **reversewrap** entry in *vtMenu.*

set-autolinefeed(*on/off/toggle*)
This action toggles automatic insertion of linefeeds and is also invoked by the **autolinefeed** entry in *vtMenu.*

set-appcursor(*on/off/toggle*)
This action toggles the handling Application Cursor Key mode and is also invoked by the Bappcursor entry in *vtMenu.*

set-appkeypad(*on/off/toggle*)
This action toggles the handling of Application Keypad mode and is also invoked by the **appkeypad** entry in *vtMenu.*

set-scroll-on-key(*on/off/toggle*)
This action toggles the **scrollKey** resource and is also invoked from the **scrollkey** entry in *vtMenu.*

set-scroll-on-tty-output(*on/off/toggle*)
This action toggles the **scrollTtyOutput** resource and is also invoked from the **scrollttyoutput** entry in *vtMenu.*

set-allow132(*on/off/toggle*)
This action toggles the **c132** resource and is also invoked from the **allow132** entry in *vtMenu.*

set-cursesemul(*on/off/toggle*)
This action toggles the **curses** resource and is also invoked from the **cursesemul** entry in *vtMenu.*

set-visual-bell(*on/off/toggle*)
: This action toggles the **visualBell** resource and is also invoked by the **visualbell** entry in *vtMenu*.

set-marginbell(*on/off/toggle*)
: This action toggles the **marginBell** resource and is also invoked from the **marginbell** entry in *vtMenu*.

set-altscreen(*on/off/toggle*)
: This action toggles between the alternative and current screens.

soft-reset()
: This action resets the scrolling region and is also invoked from the **softreset** entry in *vtMenu*.

hard-reset()
: This action resets the scrolling region, tabs, window size, and cursor keys and clears the screen. It is also invoked from the **hardreset** entry in *vtMenu*.

set-terminal-type(*type*)
: This action directs output to either the *vt* or *tek* windows, according to the *type* string. It is also invoked by the **tekmode** entry in *vtMenu* and the **vtmode** entry in *tekMenu*.

set-visibility(*vt/tek,on/off/toggle*)
: This action controls whether or not the *vt* or *tek* windows are visible. It is also invoked from the **tekshow** and **vthide** entries in *vtMenu* and the **vtshow** and **tekhide** entries in *tekMenu*.

set-tek-text(*large/2/3/small*)
: This action sets font used in the Tektronix window to the value of the resources **tektextlarge**, **tektext2**, **tektext3**, and **tektextsmall** according to the argument. It is also by the entries of the same names as the resources in *tekMenu*.

tek-page()
: This action clears the Tektronix window and is also invoked by the **tekpage** entry in *tekMenu*.

tek-reset()
: This action resets the Tektronix window and is also invoked by the *tekreset* entry in *tekMenu*.

tek-copy()
: This action copies the escape codes used to generate the current window contents to a file in the current directory beginning with the name COPY. It is also invoked from the *tekcopy* entry in *tekMenu*.

gin-press(*l/L/m/M/r/R*)
This action send the indicated graphics input code.

The default bindings in the VT102 window are:

Shift <KeyPress> Prior:	scroll-back(1,halfpage)
Shift <KeyPress> Next:	scroll-forw(1,halfpage)
Shift <KeyPress> Select:	select-cursor-start() select-cursor-end(PRIMARY, CUT_BUFFER0)
Shift <KeyPress> Insert:	insert-selection(PRIMARY, CUT_BUFFER0)
~Meta<KeyPress>:	insert-seven-bit()
Meta<KeyPress>:	insert-eight-bit()
Ctrl ~Meta<Btn1Down>:	popup-menu(mainMenu)
~Meta <Btn1Down>:	select-start()
~Meta <Btn1Motion>:	select-extend()
Ctrl ~Meta <Btn2Down>:	popup-menu(vtMenu)
~Ctrl ~Meta <Btn2Down>:	ignore()
~Ctrl ~Meta <Btn2Up>:	insert-selection(PRIMARY, CUT_BUFFER0)
Ctrl ~Meta <Btn3Down>:	popup-menu(fontMenu)
~Ctrl ~Meta <Btn3Down>:	start-extend()
~Meta <Btn3Motion>:	select-extend()
~Ctrl ~Meta <BtnUp>:	select-end(PRIMARY, CUT_BUFFER0)
<BtnDown>:	bell(0)

The default bindings in the Tektronix window are:

~Meta<KeyPress>:	insert-seven-bit()
Meta<KeyPress>:	insert-eight-bit()
Ctrl ~Meta<Btn1Down>:	popup-menu(mainMenu)
Ctrl ~Meta <Btn2Down>:	popup-menu(tekMenu)
Shift ~Meta<Btn1Down>:	gin-press(L)
~Meta<Btn1Down>:	gin-press(l)
Shift ~Meta<Btn2Down>:	gin-press(M)
~Meta<Btn2Down>:	gin-press(m)
Shift ~Meta<Btn3Down>:	gin-press(R)
~Meta<Btn3Down>:	gin-press(r)

Below is a sample how of the **keymap()** action is used to add special keys for entering commonly-typed works:

```
*VT100.Translations: #override <Key>F13: keymap(dbx)
*VT100.dbxKeymap.translations: \
  <Key>F14: keymap(None) \n\
  <Key>F17: string("next") string(0x0d) \n\
  <Key>F18: string("step") string(0x0d) \n\
  <Key>F19: string("continue") string(0x0d) \n\
  <Key>F20: string("print ") insert-selection(PRIMARY, CUT_BUFFER0)
```

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replace with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

SEE ALSO

resize(1), X(1), pty(4), tty(4)
Xterm Control Sequences

BUGS

The *Xterm Control Sequences* document has yet to be converted from X10. The old version, along with a first stab at an update, are available in the sources.

The class name is *XTerm* instead of *Xterm.*

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

The Tek widget does not support key/button re-binding.

COPYRIGHT

AUTHORS

Far too many people, including:

Loretta Guarino Reid (DEC-UEG-WSL), Joel McCormack (DEC-UEG-WSL), Terry Weissman (DEC-UEG-WSL), Edward Moy (Berkeley), Ralph R. Swick (MIT-Athena), Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT X Consortium), Doug Mink (SAO), Steve Pitschke (Stellar), Ron Newman (MIT–Athena), Jim Fulton (MIT X Consortium), Dave Serisky (HP)

NAME

xwd - dump an image of an X window

SYNOPSIS

xwd [-debug] [-help] [-nobdrs] [-out *file*] [-xy] [-frame] [-display *display*]

DESCRIPTION

Xwd is an X Window System window dumping utility. *Xwd* allows X users to store window images in a specially formatted dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing, etc. The target window is selected by clicking the mouse in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed.

OPTIONS

-display *display*
This argument allows you to specify the server to connect to; see *X(1)*.

-help Print out the 'Usage:' command syntax summary.

-nobdrs This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

-out *file* This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

-xy This option applies to color displays only. It selects 'XY' format dumping instead of the default 'Z' format.

-add *value*
This option specifies an signed value to be added to every pixel.

-frame This option indicates that the window manager frame should be included when manually selecting a window.

ENVIRONMENT

DISPLAY
To get default host and display number.

FILES

XWDFile.h
X Window Dump File format definition file.

SEE ALSO

xwud(1), xpr(1), X(1)

COPYRIGHT

AUTHORS

Tony Della Fera, Digital Equipment Corp., MIT Project Athena
William F. Wyatt, Smithsonian Astrophysical Observatory

NAME

xwininfo - window information utility for X

SYNOPSIS

xwininfo [-help] [-id *id*] [-root] [-name *name*] [-int] [-tree] [-stats] [-bits] [-events] [-size] [-wm] [-frame] [-all] [-english] [-metric] [-display *display*]

DESCRIPTION

Xwininfo is a utility for displaying information about windows. Various information is displayed depending on which options are selected. If no options are chosen, **-stats** is assumed.

The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its window id on the command line with the **-id** option. Or instead of specifying the window by its id number, the **-name** option may be used to specify which window is desired by name. There is also a special **-root** option to quickly obtain information on X's root window.

OPTIONS

-help Print out the 'Usage:' command syntax summary.

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.

-name *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

-int This option specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.

-tree This option causes the root, parent, and children windows' ids and names of the selected window to be displayed.

-stats This option causes the display of various attributes pertaining to the location and appearance of the selected window. Information displayed includes the location of the window, its width and height, its depth, border width, class, colormap id if any, map state, backing-store hint, and location of the corners.

-bits This option causes the display of various attributes pertaining to the selected window's raw bits and how the selected window is to be stored. Displayed information includes the selected window's bit gravity, window gravity, backing-store hint, backing-planes value, backing pixel, and whether or not the window has save-under set.

-events This option causes the selected window's event masks to be displayed. Both the event mask of events wanted by some client and the event mask of events not to propagate are displayed.

-size This option causes the selected window's sizing hints to be displayed. Displayed information includes: for both the normal size hints and the zoom size hints, the user supplied location if any; the program supplied location if any; the user supplied size if any; the program supplied size if any; the minimum size if any; the maximum size if any; the resize increments if any; and the minimum and maximum aspect ratios if any.

-wm This option causes the selected window's window manager hints to be displayed. Information displayed may include whether or not the application accepts input, what the window's icon window # and name is, where the window's icon should go, and what the window's initial state should be.

-frame This option causes window manager frames not be ignored when manually selecting windows.

-metric This option causes all individual height, width, and x and y positions to be displayed in millimeters as well as number of pixels, based on what the server thinks the resolution is. Geometry specifications that are in +x+y form are not changed.

-english This option causes all individual height, width, and x and y positions to be displayed in inches (and feet, yards, and miles if necessary) as well as number of pixels. **-metric** and **-english** may both be enabled at the same time.

-all This option is a quick way to ask for all information possible.

-display *display*
This option allows you to specify the server to connect to; see *X(1)*.

EXAMPLE

The following is a sample summary taken with no options specified:

```
xwininfo ==> Please select the window about which you
         ==> would like information by clicking the
```

```
          ==> mouse in that window.

     xwininfo ==> Window id: 0x60000f (xterm)

          ==> Upper left X: 4
          ==> Upper left Y: 19
          ==> Width: 726
          ==> Height: 966
          ==> Depth: 4
          ==> Border width: 3
          ==> Window class: InputOutput
          ==> Colormap: 0x80065
          ==> Window Bit Gravity State: NorthWestGravity
          ==> Window Window Gravity State: NorthWestGravity
          ==> Window Backing Store State: NotUseful
          ==> Window Save Under State: no
          ==> Window Map State: IsViewable
          ==> Window Override Redirect State: no
          ==> Corners:  +4+19  -640+19  -640-33  +4-33
```

ENVIRONMENT

DISPLAY

To get the default host and display number.

SEE ALSO

X(1), xprop(1)

BUGS

Using **-stats -bits** shows some redundant information.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xwud – image displayer for X

SYNOPSIS

xwud [-in *file*] [-noclick] [-geometry *geom*] [-display *display*] [-new] [-std <maptype>] [-raw] [-vis <vis-type-or-id>] [-help] [-rv] [-plane *number*] [-fg *color*] [-bg *color*]

DESCRIPTION

Xwud is an X Window System image undumping utility. *Xwud* allows X users to display in a window an image saved in a specially formatted dump file, such as produced by *xwd(1)*.

OPTIONS

-bg *color*
: If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "0" bits in the image.

-display *display*
: This option allows you to specify the server to connect to; see *X(1)*.

-fg *color*
: If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "1" bits in the image.

-geometry *geom*
: This option allows you to specify the size and position of the window. Typically you will only want to specify the position, and let the size default to the actual size of the image.

-help
: Print out a short description of the allowable options.

-in *file*
: This option allows the user to explicitly specify the input file on the command line. If no input file is given, the standard input is assumed.

-new
: This option forces creation of a new colormap for displaying the image. If the image characteristics happen to match those of the display, this can get the image on the screen faster, but at the cost of using a new colormap (which on most displays will cause other windows to go technicolor).

-noclick
: Clicking any button in the window will terminate the application, unless this option is specified. Termination can always be achieved by typing 'q', 'Q', or ctrl-c.

-plane *number*
You can select a single bit plane of the image to display with this option. Planes are numbered with zero being the least significant bit. This option can be used to figure out which plane to pass to *xpr(1)* for printing.

-raw This option forces the image to be displayed with whatever color values happen to currently exist on the screen. This option is mostly useful when undumping an image back onto the same screen that the image originally came from, while the original windows are still on the screen, and results in getting the image on the screen faster.

-rv If a bitmap image (or a single plane of an image) is displayed, this option forces the foreground and background colors to be swapped. This may be needed when displaying a bitmap image which has the color sense of pixel values "0" and "1" reversed from what they are on your display.

-std *maptype*
This option causes the image to be displayed using the specified Standard Colormap. The property name is obtained by converting the type to upper case, prepending "RGB_", and appending "_MAP". Typical types are "best", "default", and "gray". See *xcmap(1)* for one way of creating Standard Colormaps.

-vis *vis-type-or-id*
This option allows you to specify a particular visual or visual class. The default is to pick the "best" one. A particular class can be specified: "StaticGray", "GrayScale", "StaticColor", "PseudoColor", "DirectColor", or "TrueColor". Or "Match" can be specified, meaning use the same class as the source image. Alternatively, an exact visual id (specific to the server) can be specified, either as a hexadecimal number (prefixed with "0x") or as a decimal number. Finally, "default" can be specified, meaning to use the same class as the colormap of the root window. Case is not signficant in any of these strings.

ENVIRONMENT

DISPLAY
To get default display.

FILES

XWDFile.h
X Window Dump File format definition file.

SEE ALSO

xwd(1), xpr(1), xcmap(1), X(1)

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Bob Scheifler, MIT X Consortium

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [–vdlt] grammar

DESCRIPTION

The *yacc* command converts a context-free grammar into a set of tables for a simple automaton which executes an *LR*(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **–v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **–d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **–l** flag is given, the code produced in **y.tab.c** will *not* contain any #line constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **–t** option is used, this debugging code will be compiled by default. Independent of whether the **–t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp,	
yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1).
Programmer's Guide.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

CAVEAT

Because file names are fixed, at most one *yacc* process can be active in a given directory at a given time.